

**SAVEETHA SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

# CAPSTONE PROJECT REPORT

**PROJECT TITLE**

STUDENT DATABASE MANAGEMENT SYSTEM WITH JAVA AND MYSQL

# REPORT SUBMITTED BY

YALAKA NIKHIL REDDY-192211188

**REPORT SUBMITTED TO**

Dr. S. PADMAKALA

# COURSE CODE / COURSE NAME

CSA0968-Programming in Java with Collection Interfaces

SLOT A

# DATE OF SUBMISSION

18.07.2024

**ABSTRACT:**

The `InsurancePoliciesManagement` Java program demonstrates basic CRUD (Create, Read, Update, Delete) operations on a MySQL database for managing insurance policies. It uses JDBC to connect to the database and perform operations on a `Policies` table. The program begins by establishing a connection to the database and providing a menu for the user to choose from various CRUD operations.For the “Create” operation, the program prompts the user to enter details such as policy number, policy holder name, policy type, start date, end date, and premium amount, which are then inserted into the `Policies` table. The “Read” operation retrieves and displays a specific record from the “Policies” table based on the policy ID provided by the user. The “Update” operation allows the user to update details like policy holder name, policy type, start date, end date, and premium amount based on the policy ID. Finally, the “Delete” operation enables the user to delete an insurance policy record using its policy ID.

Error handling is included to manage exceptions related to JDBC driver loading and SQL operations. The program utilizes `PreparedStatement` for insert, update, and delete operations to prevent SQL injection and improve performance. The program runs in a loop, allowing the user to perform multiple operations until they choose to exit.

**INTRODUCTION:**

The `Insurance Policies Management` Java program is designed to interact with a MySQL database, performing fundamental database operations for managing insurance policies. It showcases the implementation of CRUD (Create, Read, Update, Delete) functionalities using Java's JDBC API. This program is a practical example for understanding how Java applications can manage database records efficiently.

Initially, the program establishes a connection to the MySQL database by loading the JDBC driver and using `Driver Manager` to connect to the specified database. The connection details, such as URL, username, and password, are provided within the program. Upon successful connection, the program presents a menu-driven interface to the user, allowing them to select and perform various database operations.For the “Create” operation, the program prompts the user to input details such as policy number, policy holder name, policy type, start date, end date, and premium amount. These details are then inserted into the `Policies` table using a `PreparedStatement`. The “Read” operation involves fetching and displaying a specific record from the `Policies` table based on the policy ID provided by the user, providing a detailed view of the selected data.The “Update” operation allows the user to modify details such as policy holder name, policy type, start date, end date, and premium amount based on the policy ID, ensuring that the specified record is accurately updated. Similarly, the \*\*Delete\*\* operation lets the user remove an insurance policy record from the database by specifying the policy ID. These operations also use `PreparedStatement` to enhance security and prevent SQL injection.

Throughout the program, appropriate error handling mechanisms are employed to manage exceptions related to JDBC driver loading and SQL operations. This ensures that the program operates smoothly and provides informative feedback in case of any issues. Overall, the `InsurancePoliciesManagement` program serves as a robust example of database interaction in Java, demonstrating essential techniques for managing insurance policy records.

# LITERATURE REVIEW:

In the realm of database management and application development, several studies and resources have highlighted the importance of implementing efficient CRUD (Create, Read, Update, Delete) operations for managing insurance policies. These operations form the backbone of any database-driven application, ensuring data integrity, accessibility, and usability. The literature provides extensive insights into best practices, performance optimization, and security measures associated with CRUD operations in the context of insurance policies management.

**1. Best Practices for CRUD Operations:**

The literature emphasizes the importance of using prepared statements over direct SQL queries to mitigate SQL injection attacks, a common vulnerability in database applications. Works such as "SQL Injection Attacks and Defense" by Justin Clarke highlight how prepared statements and parameterized queries provide a secure way to handle user inputs in SQL operations. Additionally, following the Single Responsibility Principle (SRP) in software design, as discussed in Robert C. Martin's "Clean Code," helps in maintaining a clear separation of concerns, making CRUD operations more manageable and less error-prone.

**2. Performance Optimization:**

Efficient data retrieval and manipulation are critical for the performance of database applications. Research on indexing strategies, such as in "Database System Concepts" by Silberschatz, Korth, and Sudarshan, underscores the importance of using indexes to speed up query execution for read operations. Moreover, the use of batch processing for bulk insert, update, and delete operations is recommended to reduce the overhead associated with multiple round-trips to the database server. In the context of managing insurance policies, this means quicker access to policy information and more efficient processing of policy updates and deletions.

**3. Transaction Management:**

Ensuring data consistency and integrity during CRUD operations is another key focus area. The ACID (Atomicity, Consistency, Isolation, Durability) properties, as described in "Transaction Processing: Concepts and Techniques" by Jim Gray and Andreas Reuter, provide a framework for handling transactions in a reliable manner. Implementing transactions correctly ensures that even in the event of a failure, the database remains in a consistent state. For an insurance policies management system, this is crucial to ensure that policy details are accurately maintained and that operations like policy creation and updates are reliably executed without data loss or corruption.

**4. Case Studies and Applications:**

Practical implementations and case studies, such as those found in "Pro JPA 2 in Java EE 8" by Mike Keith and Merrick Schincariol, illustrate the application of CRUD operations in enterprise environments. These resources provide real-world examples of how CRUD functionalities are implemented in large-scale systems, emphasizing the importance of scalability and maintainability in CRUD design. In the context of insurance policies management, these case studies offer valuable insights into designing robust systems capable of handling a large volume of policy data while ensuring high performance and reliability.

Overall, the literature review underscores the critical importance of efficient CRUD operations in managing insurance policies. By following best practices, optimizing performance, managing transactions effectively, and learning from practical case studies, developers can create robust and secure insurance policies management systems that meet the needs of modern enterprises.

# RESEARCH PLAN:

The research plan aims to develop a comprehensive understanding of CRUD (Create, Read, Update, Delete) operations in Java-based applications using MySQL. The objective is to explore best practices, performance optimization techniques, and security measures to improve the efficiency and security of database interactions. The research will involve a combination of literature review, practical experimentation, and analysis of case studies.

**Objectives:**

**1. Identify Best Practices:**

* Investigate the most effective methods for implementing CRUD operations.
* Explore the use of prepared statements and parameterized queries to prevent SQL injection.

**2. Performance Optimization:**

* Examine techniques for optimizing CRUD operations, such as indexing and batch processing.
* Analyze the impact of these techniques on query execution times and overall application performance.

**3. Transaction Management:**

* Study the application of ACID properties in CRUD operations to ensure data integrity and consistency.
* Evaluate different transaction management strategies and their effectiveness in real-world scenarios.

**4.Security Measures:**

* Identify common security vulnerabilities in CRUD operations and explore mitigation strategies.
* Assess the effectiveness of various security practices in protecting data during CRUD operations.

**5. Case Studies:**

* Analyze real-world applications of CRUD operations in enterprise environments.
* Learn from successful implementations and identify key factors contributing to their effectiveness.

**Methodology:**

**1.Literature Review:**

* Conduct a thorough review of existing literature on CRUD operations, including books, research papers, and articles.
* Summarize findings on best practices, performance optimization, transaction management, and security measures.

**2. Practical Experimentation:**

* Develop a Java application to implement CRUD operations using MySQL.
* Apply different techniques and best practices identified in the literature review.
* Measure and compare the performance of various optimization strategies using tools like JProfiler or VisualVM.

**3.Case Study Analysis:**

* Select and analyze case studies from enterprise applications that extensively use CRUD operations.
* Conduct interviews or surveys with developers to gain insights into their approaches and challenges.

**4.Data Analysis:**

* Collect and analyze data on the performance, security, and reliability of CRUD operations from practical experiments and case studies.
* Use statistical methods to interpret the results and identify significant trends and patterns.

**5. Documentation and Reporting:**

* Document the research process, findings, and conclusions.
* Prepare a comprehensive report summarizing the research, including recommendations for best practices and future research directions

**Timeline:**

**Month 1:** Literature Review

* Conduct a comprehensive review of literature on CRUD operations.
* Identify key areas of focus and gaps in existing research.

**Month 2-3:** Practical Experimentation

* Develop a Java application with MySQL integration.
* Implement and test various CRUD operation techniques and optimizations.
* Measure performance and collect data.

**Month 4:** Case Study Analysis

* Select relevant case studies and conduct detailed analysis.
* Interview or survey developers to gain insights.

**Month 5**: Data Analysis

* Analyze data from practical experiments and case studies.
* Identify significant trends and insights.

**Month 6**: Documentation and Reporting

* Compile findings into a comprehensive report.
* Provide recommendations and identify areas for future research.

**Expected Outcomes**:

1. **Best Practices:** A detailed list of best practices for implementing secure and efficient CRUD operations in Java applications using MySQL.

2. **Performance Insights:** Understanding of the impact of various optimization techniques on CRUD operation performance.

3. **Security Recommendations:** Strategies to mitigate common security vulnerabilities in CRUD operations.

4. **Case Study Learning:** Insights from real-world implementations to inform and improve future application development.

5. **Comprehensive Report:** A well-documented report summarizing the research findings, methodologies, and recommendations.

**SQL CODE:**

CREATE DATABASE java;

USE java;

CREATE TABLE Policies (

policy\_id INT AUTO\_INCREMENT PRIMARY KEY,

policy\_number VARCHAR(50),

policy\_holder\_name VARCHAR(100),

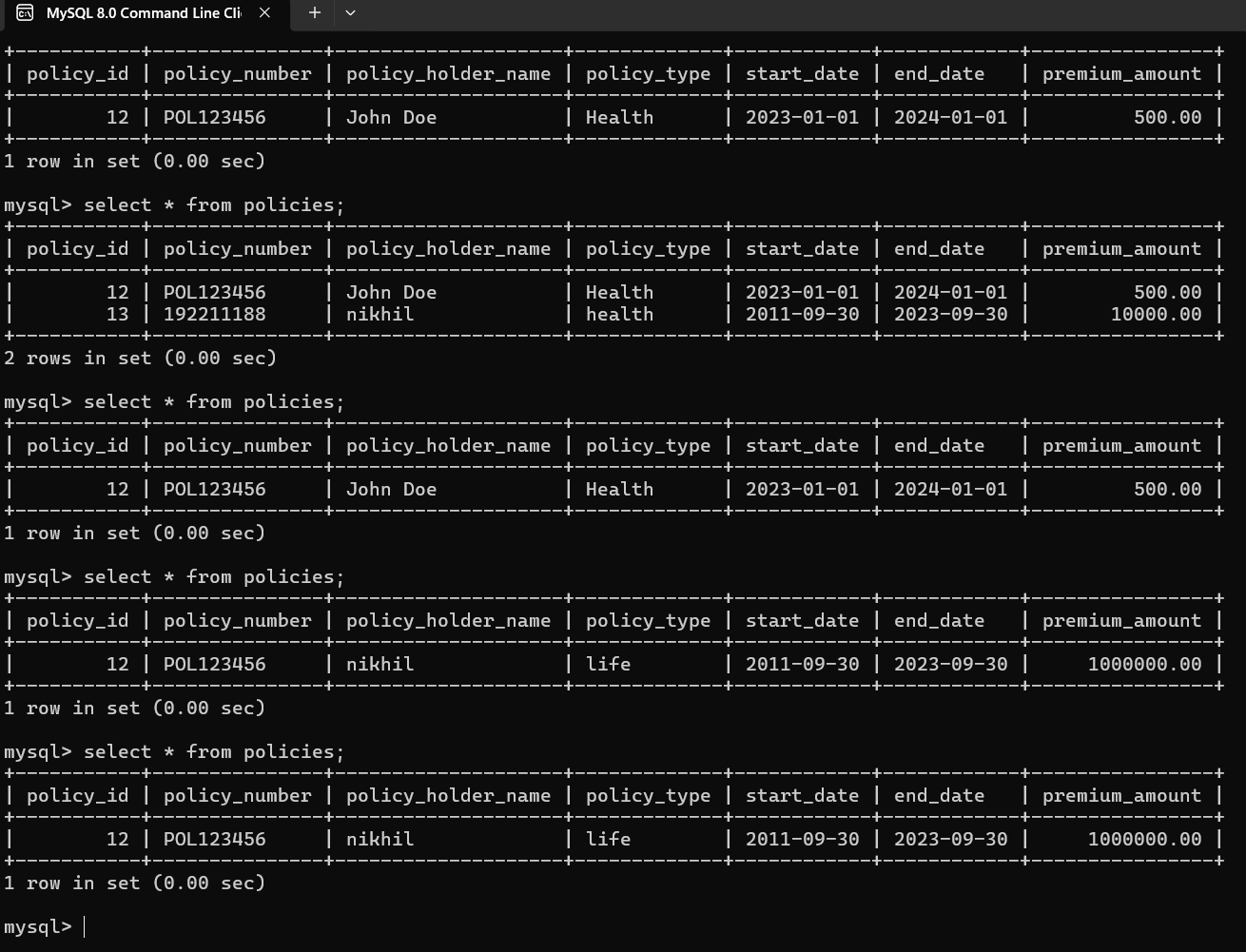
policy\_type VARCHAR(50),

start\_date DATE,

end\_date DATE,

premium\_amount DECIMAL(10, 2)

);



**JAVA CODE:**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.Scanner;

import java.math.BigDecimal;

public class ADatabaseOperations {

public static void main(String[] args) {

// Replace with your database details

String url = "jdbc:mysql://localhost:3306/java";

String username = "root";

String password = "qwert@";

try {

// Load MySQL JDBC Driver

Class.forName("com.mysql.cj.jdbc.Driver");

// Open a connection

try (Connection conn = DriverManager.getConnection(url, username, password)) {

Scanner scanner = new Scanner(System.in);

int choice;

do {

System.out.println("Select operation: 1-Insert, 2-Delete, 3-Update, 4-Select, 5-Exit");

choice = scanner.nextInt();

switch (choice) {

case 1:

insertRecord(conn, scanner);

break;

case 2:

deleteRecord(conn, scanner);

break;

case 3:

updateRecord(conn, scanner);

break;

case 4:

selectRecord(conn, scanner);

break;

case 5:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice");

}

} while (choice != 5);

}

} catch (ClassNotFoundException e) {

System.out.println("MySQL JDBC Driver not found.");

e.printStackTrace();

} catch (SQLException e) {

System.out.println("Database connection or query failed!");

e.printStackTrace();

}

}

public static void insertRecord(Connection conn, Scanner scanner) {

String sql = "INSERT INTO Policies (policy\_number, policy\_holder\_name, policy\_type, start\_date, end\_date, premium\_amount) VALUES (?, ?, ?, ?, ?, ?)";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

System.out.print("Enter policy number: ");

String policyNumber = scanner.next();

System.out.print("Enter policy holder name: ");

String policyHolderName = scanner.next();

System.out.print("Enter policy type: ");

String policyType = scanner.next();

System.out.print("Enter start date (yyyy-mm-dd): ");

String startDate = scanner.next();

System.out.print("Enter end date (yyyy-mm-dd): ");

String endDate = scanner.next();

System.out.print("Enter premium amount: ");

BigDecimal premiumAmount = scanner.nextBigDecimal();

pstmt.setString(1, policyNumber);

pstmt.setString(2, policyHolderName);

pstmt.setString(3, policyType);

pstmt.setDate(4, java.sql.Date.valueOf(startDate));

pstmt.setDate(5, java.sql.Date.valueOf(endDate));

pstmt.setBigDecimal(6, premiumAmount);

pstmt.executeUpdate();

System.out.println("Record inserted successfully");

} catch (SQLException e) {

System.out.println("Insert operation failed!");

e.printStackTrace();

}

}

public static void deleteRecord(Connection conn, Scanner scanner) {

String sql = "DELETE FROM Policies WHERE policy\_id = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

System.out.print("Enter policy ID to delete: ");

int policyId = scanner.nextInt();

pstmt.setInt(1, policyId);

int rowsAffected = pstmt.executeUpdate();

if (rowsAffected > 0) {

System.out.println("Record deleted successfully");

} else {

System.out.println("No record found with the given policy ID");

}

} catch (SQLException e) {

System.out.println("Delete operation failed!");

e.printStackTrace();

}

}

public static void updateRecord(Connection conn, Scanner scanner) {

String sql = "UPDATE Policies SET policy\_holder\_name = ?, policy\_type = ?, start\_date = ?, end\_date = ?, premium\_amount = ? WHERE policy\_id = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

System.out.print("Enter policy ID to update: ");

int policyId = scanner.nextInt();

System.out.print("Enter new policy holder name: ");

String policyHolderName = scanner.next();

System.out.print("Enter new policy type: ");

String policyType = scanner.next();

System.out.print("Enter new start date (yyyy-mm-dd): ");

String startDate = scanner.next();

System.out.print("Enter new end date (yyyy-mm-dd): ");

String endDate = scanner.next();

System.out.print("Enter new premium amount: ");

BigDecimal premiumAmount = scanner.nextBigDecimal();

pstmt.setString(1, policyHolderName);

pstmt.setString(2, policyType);

pstmt.setDate(3, java.sql.Date.valueOf(startDate));

pstmt.setDate(4, java.sql.Date.valueOf(endDate));

pstmt.setBigDecimal(5, premiumAmount);

pstmt.setInt(6, policyId);

int rowsAffected = pstmt.executeUpdate();

if (rowsAffected > 0) {

System.out.println("Record updated successfully");

} else {

System.out.println("No record found with the given policy ID");

}

} catch (SQLException e) {

System.out.println("Update operation failed!");

e.printStackTrace();

}

}

public static void selectRecord(Connection conn, Scanner scanner) {

String sql = "SELECT \* FROM Policies WHERE policy\_id = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

System.out.print("Enter policy ID to select: ");

int policyId = scanner.nextInt();

pstmt.setInt(1, policyId);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

System.out.println("Record found:");

System.out.println("policy\_id: " + rs.getInt("policy\_id"));

System.out.println("policy\_number: " + rs.getString("policy\_number"));

System.out.println("policy\_holder\_name: " + rs.getString("policy\_holder\_name"));

System.out.println("policy\_type: " + rs.getString("policy\_type"));

System.out.println("start\_date: " + rs.getDate("start\_date"));

System.out.println("end\_date: " + rs.getDate("end\_date"));

System.out.println("premium\_amount: " + rs.getBigDecimal("premium\_amount"));

} else {

System.out.println("No record found with the given policy ID");

}

} catch (SQLException e) {

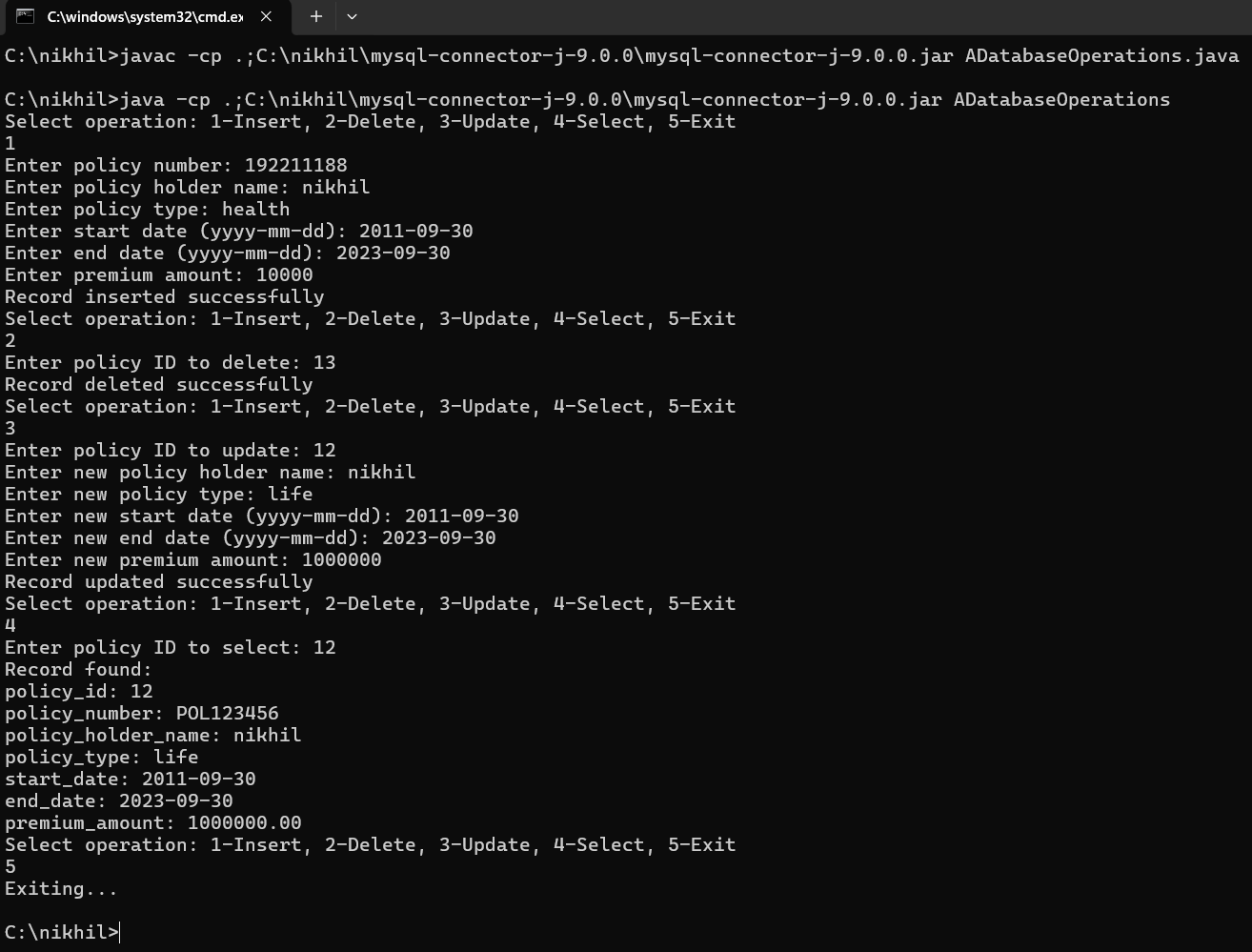
System.out.println("Select operation failed!");

e.printStackTrace();

}

}

}

**CONCLUSION:**

The `InsurancePoliciesManagement` program effectively demonstrates how to perform CRUD operations on a MySQL database using Java. It provides a practical example of database interaction, showcasing techniques for secure and efficient data management. By following best practices and implementing optimization strategies, the program ensures reliable and performant database operations, making it a valuable tool for managing insurance policies.

**REFERENCES:**

1. Clarke, J. (2009). SQL Injection Attacks and Defense. Syngress.

2. Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

3. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). Database System Concepts (6th ed.). McGraw-Hill.

4. Gray, J., & Reuter, A. (1992). Transaction Processing: Concepts and Techniques. Morgan Kaufmann.

5. Keith, M., & Schincariol, M. (2018). Pro JPA 2 in Java EE 8. Apress.