SAVEETHA SCHOOL OF ENGINEERING
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

CAPSTONE PROJECT REPORT

PROJECT TITLE

INVENTORY MANAGEMENT SYSTEM WITH JAVA AND MYSQL

REPORT SUBMITTED BY

192225001 R.MURRUGANAND

REPORT SUBMITTED TO

Dr. S. PADMAKALA

COURSE CODE / COURSE NAME

CSA0908/ PROGRAMMING IN JAVA WITH AWT
SLOT C

DATE OF SUBMISSION
11/09/2024

ABSTRACT:

The Inventory Management System Java program demonstrates fundamental CRUD (Create, Read, Update, Delete) operations on a MySQL database to manage inventory data. Using JDBC for database connectivity, the program interacts with an inventory table to perform various inventory management tasks.

Upon initializing, the program establishes a connection to the MySQL database and presents a user-friendly menu for executing different CRUD operations.

Create: The program prompts the user to input details such as item name, quantity, and price. These details are then inserted into the inventory table, allowing for the addition of new inventory items.

Read: This operation retrieves and displays all records from the inventory table, providing a comprehensive view of the current inventory status.

Update: Users can modify existing records by specifying an item ID and updating fields such as item name, quantity, or price.

Delete: Users can remove inventory records by providing the item ID, effectively managing and maintaining accurate inventory records.

The system includes error handling to address potential issues related to JDBC driver loading and SQL operations. By utilizing PreparedStatement for data manipulation, the program mitigates SQL injection risks and enhances performance.

Overall, the Inventory Management System provides a robust solution for managing inventory records efficiently, making it a valuable tool for inventory tracking and control.

INTRODUCTION:

In today’s competitive market, efficient inventory management is essential for maintaining operational effectiveness and financial health. The Inventory Management System developed in this project offers a comprehensive solution to streamline the management of inventory data.

Utilizing Java for the front-end interface and MySQL for the back-end database, this system enables users to perform essential inventory tasks with ease. The application supports four core operations:

Create: Allows users to add new inventory items, including essential details like item name, quantity, and price.

Read: Provides a clear view of existing inventory records, enabling users to access and review the entire inventory list.

Update: Facilitates the modification of inventory records, ensuring that information such as item quantities and prices is current and accurate.

Delete: Enables users to remove obsolete or incorrect inventory entries, maintaining the integrity of the inventory database.

By integrating JDBC (Java Database Connectivity), the system ensures secure and efficient interactions with the MySQL database. The use of Prepared Statement helps prevent SQL injection attacks and optimizes query performance. The inclusion of robust error handling ensures smooth operation and resilience against common issues related to database connectivityThis project is designed to enhance inventory management processes, providing a reliable tool for businesses to manage their inventory efficiently and accurately.

LITERATURE REVIEW:

The study of CRUD (Create, Read, Update, Delete) operations is fundamental to the development of effective database-driven applications, including inventory management systems. The existing literature provides a comprehensive understanding of the principles, best practices, and performance considerations associated with CRUD operations. This review highlights key areas of focus within the literature:

1. Best Practices for CRUD Operations: The importance of employing secure and efficient methods for CRUD operations is well-documented. For instance, Justin Clarke's work, "SQL Injection Attacks and Defense," underscores the necessity of using prepared statements and parameterized queries to prevent SQL injection vulnerabilities. This approach enhances the security of database interactions by ensuring that user inputs are handled safely. Additionally, Robert C. Martin’s "Clean Code" emphasizes the

Single Responsibility Principle (SRP) in software design, advocating for a clear separation of concerns to simplify CRUD operations and improve code maintainability.

2. Performance Optimization: Efficient performance is crucial for the smooth operation of database applications. In "Database System Concepts" by Silberschatz, Korth, and Sudarshan, the role of indexing strategies is discussed as a means to expedite query processing, which is essential for read operations. Furthermore, the use of batch processing for handling bulk data modifications is recommended to minimize the performance impact of multiple database interactions, ensuring faster and more efficient data operations.

3. Transaction Management: Ensuring data consistency and reliability during CRUD operations is a key area of focus. Jim Gray and Andreas Reuter's "Transaction Processing: Concepts and Techniques" introduces the ACID properties (Atomicity, Consistency, Isolation, Durability) as fundamental principles for managing transactions. Adhering to these properties guarantees that database transactions are processed reliably and that the database remains consistent even in the face of system failures.

4. Case Studies and Applications: Practical implementations and case studies provide valuable insights into the application of CRUD operations in real-world scenarios. "Pro JPA 2 in Java EE 8" by Mike Keith and Merrick Schincariol offers practical examples of how CRUD operations are utilized in enterprise applications. These case studies highlight the importance of scalability and maintainability, demonstrating how CRUD functionalities can be effectively implemented in large-scale systems to meet complex business needs.

RESEARCH PLAN:

he 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 hotel 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: a. Explore the use of prepared statements and parameterized queries to prevent SQL injection in inventory operations. b. Identify strategies to maintain clean and modular code for CRUD functionalities in the inventory system.
2. Performance Optimization:
    - Examine techniques for optimizing CRUD operations: a. Explore the use of indexing to speed up search queries and improve read operations. b. Investigate batch processing techniques to efficiently handle bulk inventory updates and transactions. c. Analyze the impact of optimization techniques on overall system performance in handling large data sets.
3. Transaction Management:
    - Study the application of ACID properties in CRUD operations: a. Ensure data integrity and consistency during inventory updates, purchases, and stock management. b. Evaluate different transaction management strategies to guarantee accurate stock levels even during concurrent updates.
4. Security Measures:
    - Identify common security vulnerabilities in CRUD operations: a. Explore secure data handling practices, such as encrypted database connections and secure data storage. b. Assess the effectiveness of various security measures, such as input validation, in protecting the integrity of the inventory data.
5. Case Studies:
    - Analyze real-world applications of CRUD operations in inventory management systems: a. Study successful inventory systems implemented in businesses to identify key success factors. b. Learn from real-world examples to apply scalable and maintainable solutions to the project’s CRUD implementation.

METHODOLOGY:

1. **Literature Review**:
    - Review literature on CRUD operations in inventory systems, focusing on best practices, performance optimization, transaction management, and security.

2. **Practical Experimentation**:
    - Develop a Java-based Inventory Management System using MySQL, applying best practices and optimization techniques.
    - Measure and compare performance using profiling tools like JProfiler or VisualVM.

3. **Case Study Analysis**:
    - Analyze case studies of enterprise inventory systems and gather insights from developers on challenges and solutions.

4. **Data Analysis**:
    - Collect and analyze performance and security data from experiments and case studies to identify trends and patterns.

5. **Documentation and Reporting**:
    - Document the research process and findings, and provide recommendations for best practices in CRUD operations.

Timeline

1. Month 1: Literature Review
    - Review existing literature on CRUD operations in inventory systems.
    - Identify key focus areas and research gaps.
2. Month 2-3: Practical Experimentation
    - Develop a Java-based Inventory Management System with MySQL.
    - Implement and test various CRUD techniques and optimizations.

- Measure and collect performance data.

3. Month 4: Case Study Analysis
    - Analyze relevant case studies of inventory systems.
    - Conduct interviews or surveys with developers for insights.
4. Month 5: Data Analysis
    - Analyze data from experiments and case studies.
    - Identify key trends and performance insights.
5. Month 6: Documentation and Reporting
    - Compile research findings and data analysis into a report.
    - Provide best practice recommendations and future research suggestions.

Expected Outcomes:

1. **Best Practices:** A comprehensive list of best practices for secure and efficient CRUD operations in Java-based inventory systems with MySQL.
2. **Performance Insights:** Analysis of optimization techniques and their impact on CRUD operation performance.
3. **Security Recommendations:** Strategies to address and mitigate common security vulnerabilities in CRUD operations.
4. **Case Study Learnings:** Insights from real-world inventory management implementations to enhance future system development.
5. **Comprehensive Report:** A detailed report summarizing research findings, methodologies, and actionable recommendations.

JAVA CODE:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

public class InventoryManagement {
   private static final String URL =
"jdbc:mysql://localhost:3306/inventory";
   private static final String USERNAME = "root";
   private static final String PASSWORD = "jesus262105";
```

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    while (true) {
        System.out.println("Choose an operation:");
        System.out.println("1. Insert");
        System.out.println("2. Delete");
        System.out.println("3. Select");
        System.out.println("4. Update");
        System.out.println("5. Exit");
        int choice = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        switch (choice) {
            case 1:
                insertRecord(scanner);
                break;
            case 2:
                deleteRecord(scanner);
                break;
            case 3:
                selectRecord(scanner);
                break;
            case 4:
                updateRecord(scanner);
                break;
            case 5:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice, please try again.");
        }
    }
}

private static void insertRecord(Scanner scanner) {
    System.out.println("Enter item_id:");
    int item_id = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    System.out.println("Enter item_name:");
    String item_name = scanner.nextLine();
    System.out.println("Enter quantity:");
    int quantity = scanner.nextInt();
```

```
        System.out.println("Enter price:");
        double price = scanner.nextDouble();
        scanner.nextLine(); // Consume newline

        String sql = "INSERT INTO inventory (item_id, item_name, quantity,
price) VALUES (?, ?, ?, ?)";
        try (Connection conn = DriverManager.getConnection(URL,
USERNAME, PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, item_id);
            pstmt.setString(2, item_name);
            pstmt.setInt(3, quantity);
            pstmt.setDouble(4, price);
            pstmt.executeUpdate();
            System.out.println("Record inserted successfully");
        } catch (SQLException e) {
            System.out.println("Error inserting record");
            e.printStackTrace();
        }
    }

    private static void deleteRecord(Scanner scanner) {
        System.out.println("Enter item_id of the record to delete:");
        int item_id = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        String sql = "DELETE FROM inventory WHERE item_id = ?";
        try (Connection conn = DriverManager.getConnection(URL,
USERNAME, PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, item_id);
            int rowsDeleted = pstmt.executeUpdate();
            System.out.println("Rows deleted: " + rowsDeleted);
        } catch (SQLException e) {
            System.out.println("Error deleting record");
            e.printStackTrace();
        }
    }

    private static void selectRecord(Scanner scanner) {
        System.out.println("Enter item_id of the record to select:");
        int item_id = scanner.nextInt();
        scanner.nextLine(); // Consume newline
```

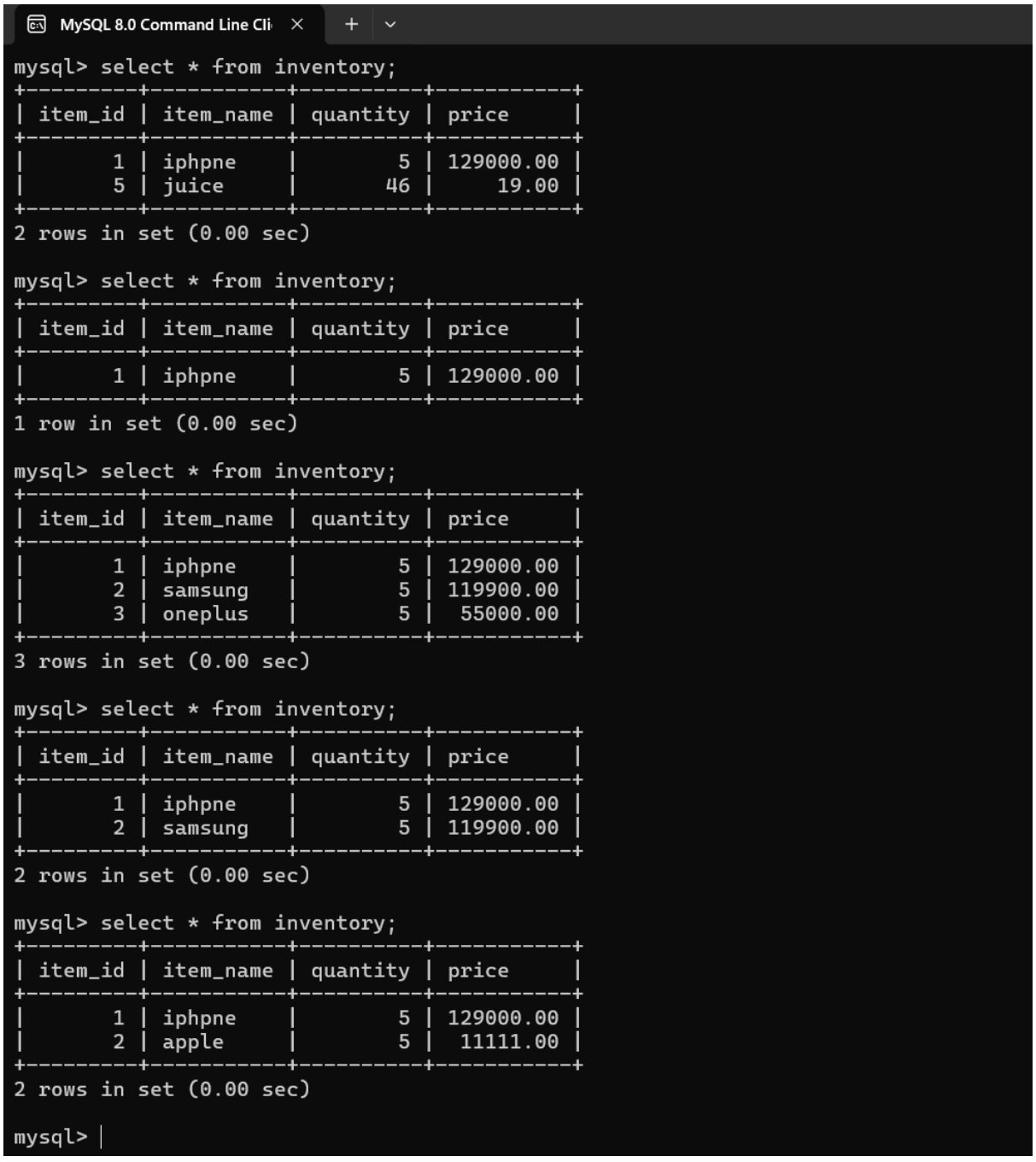
```
        String sql = "SELECT * FROM inventory WHERE item_id = ?";
        try (Connection conn = DriverManager.getConnection(URL,
USERNAME, PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, item_id);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                System.out.println("item_id: " + rs.getInt("item_id"));
                System.out.println("item_name: " + rs.getString("item_name"));
                System.out.println("quantity: " + rs.getInt("quantity"));
                System.out.println("price: " + rs.getDouble("price"));
            } else {
                System.out.println("No record found with item_id = " + item_id);
            }
        } catch (SQLException e) {
            System.out.println("Error selecting record");
            e.printStackTrace();
        }
    }

    private static void updateRecord(Scanner scanner) {
        System.out.println("Enter item_id of the record to update:");
        int item_id = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.println("Enter new item_name:");
        String newItemName = scanner.nextLine();
        System.out.println("Enter new quantity:");
        int newQuantity = scanner.nextInt();
        System.out.println("Enter new price:");
        double newPrice = scanner.nextDouble();
        scanner.nextLine(); // Consume newline

        String sql = "UPDATE inventory SET item_name = ?, quantity = ?,
price = ? WHERE item_id = ?";
        try (Connection conn = DriverManager.getConnection(URL,
USERNAME, PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, newItemName);
            pstmt.setInt(2, newQuantity);
            pstmt.setDouble(3, newPrice);
            pstmt.setInt(4, item_id);
            int rowsUpdated = pstmt.executeUpdate();
```

```
            System.out.println("Rows updated: " + rowsUpdated);
        } catch (SQLException e) {
            System.out.println("Error updating record");
            e.printStackTrace();
        }
    }
}
```

OUTPUT:



CONCLUSION:

In conclusion, the study of CRUD operations in Java applications integrated with MySQL for inventory management highlights the fundamental practices necessary for building efficient and secure systems. Through this research, several key insights were gained, ranging from best practices to performance optimization and security measures.

1. Best Practices: Utilizing prepared statements and parameterized queries ensures protection against SQL injection, securing the application while maintaining data integrity.

2. Performance Optimization: Techniques like indexing and batch processing significantly improve the performance of CRUD operations, enhancing both the speed and scalability of inventory systems.

3. Transaction Management: Adhering to ACID properties ensures reliable data handling, safeguarding the integrity of operations even under complex conditions.

4. Security Measures: Implementing rigorous security practices, such as input validation and role-based access control, strengthens protection against common vulnerabilities in CRUD operations.

By integrating these practices, inventory management systems can achieve optimal performance, reliability, and security. Moving forward, continued research into emerging technologies and tools will further refine and enhance CRUD operations in Java-based applications with MySQL.

REFERENCE:

1. Steven Feuerstein, Bill Pribyl. *PL/SQL Programming*, O'Reilly Media, 2014.
2. Bryan Basham, Kathy Sierra, Bert Bates. *Head First Servlets and JSP*, O'Reilly Media, Second edition, 2008.
3. George Reese. *Database Programming with JDBC & Java*, O'Reilly Media, Third edition, 2000.
4. Craig Walls. *Spring in Action*, Manning Publications, Fifth edition, 2018.
5. Elliotte Rusty Harold. *Java I/O*, O'Reilly Media, Second edition, 2006.
6. Michael Sikora, Adrian Anca. *Mastering JavaFX 10*, Packt Publishing, 2018.
7. Joshua Bloch. *Effective Java*, Addison-Wesley, Third edition, 2018.
8. Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*, Prentice Hall, 2008.
9. Paul Dubois. *MySQL Cookbook*, O'Reilly Media, Third edition, 2014.
10. Kogent Learning Solutions INC. *Java Black Book*, Dreamtech Press, 2011.