

**SAVEETHA SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

# CAPSTONE PROJECT REPORT

**PROJECT TITLE**

ONLINE BANKING SYSTEM WITH JAVA AND MYSQL

# REPORT SUBMITTED BY

192224188 B. HARSHA VARDHAN

**REPORT SUBMITTED TO**

Dr. S. PADMAKALA

# COURSE CODE / COURSE NAME

CSA0908/ PROGRAMMING IN JAVA WITH AWT

SLOT A

# DATE OF SUBMISSION

11.09.2024

**ABSTRACT:**

The Online Banking System is a secure, user-friendly platform developed using Java and SQL to simulate basic banking operations for users. The system allows users to create accounts, manage their finances, and track their transaction history. The primary objective is to provide an efficient interface for common banking tasks such as balance inquiries, fund transfers, deposits, withdrawals, and loan requests. Additionally, it includes an admin panel for monitoring and approving loan requests.

The system is backed by a structured relational database (SQL), which maintains information on users, accounts, transactions, and loans. The database consists of well-defined tables including Users, Accounts, Transactions, and Loans that store data efficiently and support necessary operations actions.

Java serves as the core programming language for building the system, handling user interaction and executing the business logic. The system makes use of JDBC (Java Database Connectivity) to interact with the SQL database, enabling real-time data processing and storage. Through a simple and interactive console interface, users can register, log in, perform transactions, view their account balances, and apply for loans. The admin interface allows administrators to manage user accounts and approve or reject loan requests.

This project demonstrates the practical integration of Java with SQL, providing a solid foundation for real-world banking applications. It emphasizes key aspects such as data security, transaction tracking, and efficient database management, offering a scalable solution for a modern banking environment.

**INTRODUCTION:**

The Online Banking System is a software application developed using Java and SQL to simulate and manage basic banking operations. This system enables users to perform essential banking activities such as creating accounts, managing their finances, transferring funds, and applying for loans in a secure and user-friendly environment. It is designed to streamline financial processes by providing users with an efficient platform for their banking needs, while also offering an administrative interface for managing user activities and loan approvals.

The system leverages Java as the core programming language to handle business logic, user interaction, and data processing. It integrates with a relational database (MySQL) through JDBC (Java Database Connectivity), ensuring the efficient storage and retrieval of data. The structured database is used to store information about users, accounts, transactions, and loans, allowing for seamless CRUD (Create, Read, Update, Delete) operations.

This project emphasizes security and reliability by implementing best practices such as using Prepared Statement to prevent SQL injection, ensuring that user data and transactions are securely processed. The system also features an administrative panel where authorized personnel can monitor user activities, review loan requests, and manage banking operations effectively.

By combining the power of Java for backend logic with SQL for data management, the Online Banking System serves as a scalable solution for real-world banking scenarios, making it a comprehensive project for demonstrating Java and SQL integration in a financial domain.

# LITERATURE REVIEW:

The development of online banking systems involves addressing several key areas to ensure the creation of robust, efficient, and reliable applications. This literature survey explores best practices, performance optimization techniques, transaction management strategies, and practical case studies relevant to online banking systems, particularly those implemented using Java and SQL.

**1. Best Practices for Online Banking System:** As highlighted in Clarke's "SQL Injection Attacks and Defence," using prepared statements and parameterized queries is crucial for preventing SQL injection attacks. This approach ensures that user inputs are safely handled and reduces the risk of malicious data breaches. Furthermore, adherence to the Single Responsibility Principle (SRP), discussed in Martin's "Clean Code," promotes modular design, which enhances code readability and simplifies maintenance by ensuring that each component or class has a specific, well-defined role.

**2. Performance Optimization:** Research such as "Database System Concepts" by Silberschatz, Korth, and Sudarshan emphasizes the importance of indexing to improve query performance. Indexes on frequently queried fields, such as account numbers or transaction IDs, significantly reduce the time required for data retrieval, which is crucial for operations like balance checks and transaction history queries.

**3. Transaction Management:** Implementing effective concurrency control mechanisms is crucial for managing simultaneous transactions and preventing issues such as deadlocks or data anomalies. Techniques like optimistic and pessimistic concurrency control are widely discussed in transaction management literature.

**4. Case Studies and Applications:** Research into large-scale banking systems highlights the importance of designing systems that can scale to accommodate growing numbers of users and transactions. Case studies demonstrate how to address challenges related to performance, data management, and system reliability.



# RESEARCH PLAN:

He researches plan aims to develop a comprehensive understanding of Banking 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:**
   1. Investigate the most effective methods for implementing Banking operations.
   2. Explore the use of prepared statements and parameterized queries to prevent SQL injection.
2. **2.Performance Optimization:**
   1. Examine techniques for optimizing Banking operations, such as indexing and batch processing.
   2. Analyze the impact of these techniques on query execution times and overall application performance.
3. **Transaction Management:**
   1. Study the application of ACID properties in Banking operations to ensure data integrity and consistency.
   2. Evaluate different transaction management strategies and their effectiveness in real-world scenarios.
4. **Security Measures:**
   1. Identify common security vulnerabilities in Banking operations and explore mitigation strategies.
   2. Assess the effectiveness of various security practices in protecting data during Banking operations.
5. **Case Studies:**
   1. Analyze real-world applications of Banking operations in enterprise environments.
   2. Learn from successful implementations and identify key factors contributing to their effectiveness.

### **Methodology:**

### Literature Review: Conduct a thorough review of academic papers, books, and industry reports related to Banking operations, performance optimization, transaction management, and security in online banking systems.

**Practical Experimentation**: Develop and test components of the banking system to implement and evaluate various techniques and practices.

**Case Study Analysis**: Examine real-world examples of online banking systems to learn from successful implementations and identify effective strategies.

**Data Analysis:** Collect and analyse data on the performance, security, and reliability of Online Banking System operations from practical experiments and case studies.

Use statistical methods to interpret the results and identify significant trends and patterns.

**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**

1. **Month 1: Literature Review**
   * Conduct a comprehensive review of literature on Online Banking System operations.
   * Identify key areas of focus and gaps in existing research.
2. **Month 2-3: Practical Experimentation**
   * Develop a Java application with MySQL integration.
   * Implement and test various Banking operation techniques and optimizations.
   * Measure performance and collect data.
3. **Month 4: Case Study Analysis**
   * Select relevant case studies and conduct detailed analysis.
   * Interview or survey developers to gain insights.
4. **Month 5: Data Analysis**
   * Analyse data from practical experiments and case studies.
   * Identify significant trends and insights.
5. **Month 6: Documentation and Reporting**
   * Compile findings into a comprehensive report.
   * Provide recommendations and identify areas for future research.
6. Bottom of Form

### **Expected Outcomes:**

**Identification of Best Practices**: A clear set of best practices for implementing Banking operations, securing transactions, and managing data in online banking systems.

**Performance Insights**: Understanding of optimization techniques and their impact on system performance.

**Effective Transaction Management**: Strategies for ensuring data integrity and consistency in financial transactions.

**Robust Security Measures**: Effective practices for protecting sensitive data and preventing security breaches.

**Real-World Insights**: Lessons from successful implementations to guide future development and enhance system design.

**JAVA CODE:**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.sql.ResultSet;

import java.util.Scanner;

public class BankingSystem {

private static final String url = "jdbc:mysql://localhost:3306/accounts";

private static final String username = "root";

private static final String password = "0077";

public static void main(String[] args) throws ClassNotFoundException, SQLException {

try {

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

} catch (ClassNotFoundException e) {

System.out.println(e.getMessage());

}

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

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nONLINE BANKING SYSTEM");

System.out.println("1. INSERT (CREATE ACCOUNT)");

System.out.println("2. DELETE (CLOSE ACCOUNT)");

System.out.println("3. SELECT (VIEW ACCOUNTS)");

System.out.println("4. UPDATE (UPDATE ACCOUNT DETAILS)");

System.out.println("5. EXIT");

System.out.print("Choose an option: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

createAccount(connection, scanner);

break;

case 2:

deleteAccount(connection, scanner);

break;

case 3:

viewAccounts(connection);

break;

case 4:

updateAccount(connection, scanner);

break;

case 5:

exit();

scanner.close();

return;

default:

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

}

}

} catch (SQLException | InterruptedException e) {

e.printStackTrace();

}

}

private static void createAccount(Connection connection, Scanner scanner) {

try {

System.out.print("Enter Customer Name: ");

String customerName = scanner.next();

scanner.nextLine();

System.out.print("Enter Account Number: ");

int accountNumber = scanner.nextInt();

System.out.print("Enter Address: ");

String address = scanner.next();

String sql = "INSERT INTO accounts (customer\_name, account\_number, address) " +

"VALUES ('" + customerName + "', " + accountNumber + ", '" + address + "')";

try (Statement statement = connection.createStatement()) {

int affectedRows = statement.executeUpdate(sql);

if (affectedRows > 0) {

System.out.println("Account created successfully!");

} else {

System.out.println("Account creation failed.");

}

}

} catch (SQLException e) {

e.printStackTrace();

}

}

private static void deleteAccount(Connection connection, Scanner scanner) {

try {

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

int accountId = scanner.nextInt();

if (!accountExists(connection, accountId)) {

System.out.println("Account not found for the given ID.");

return;

}

String sql = "DELETE FROM accounts WHERE account\_id = " + accountId;

try (Statement statement = connection.createStatement()) {

int affectedRows = statement.executeUpdate(sql);

if (affectedRows > 0) {

System.out.println("Account deleted successfully!");

} else {

System.out.println("Account deletion failed.");

}

}

} catch (SQLException e) {

e.printStackTrace();

}

}

private static void viewAccounts(Connection connection) throws SQLException {

String sql = "SELECT account\_id, customer\_name, account\_number, address, creation\_date FROM accounts";

try (Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(sql)) {

System.out.println("Current Accounts:");

System.out.println("+------------+-----------------+----------------+-------------------+-------------------------+");

System.out.println("| Account ID | Customer Name | Account Number | Address | Creation Date |");

System.out.println("+------------+-----------------+----------------+-------------------+-------------------------+");

while (resultSet.next()) {

int accountId = resultSet.getInt("account\_id");

String customerName = resultSet.getString("customer\_name");

int accountNumber = resultSet.getInt("account\_number");

String address = resultSet.getString("address");

String creationDate = resultSet.getTimestamp("creation\_date").toString();

System.out.printf("| %-10d | %-15s | %-14d | %-17s | %-23s |\n",

accountId, customerName, accountNumber, address, creationDate);

}

System.out.println("+------------+-----------------+----------------+-------------------+-------------------------+");

}

}

private static void updateAccount(Connection connection, Scanner scanner) {

try {

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

int accountId = scanner.nextInt();

scanner.nextLine();

if (!accountExists(connection, accountId)) {

System.out.println("Account not found for the given ID.");

return;

}

System.out.print("Enter new Customer Name: ");

String newCustomerName = scanner.nextLine();

System.out.print("Enter new Account Number: ");

int newAccountNumber = scanner.nextInt();

System.out.print("Enter new Address: ");

String newAddress = scanner.next();

String sql = "UPDATE accounts SET customer\_name = '" + newCustomerName + "', " +

"account\_number = " + newAccountNumber + ", " +

"address = '" + newAddress + "' " +

"WHERE account\_id = " + accountId;

try (Statement statement = connection.createStatement()) {

int affectedRows = statement.executeUpdate(sql);

if (affectedRows > 0) {

System.out.println("Account updated successfully!");

} else {

System.out.println("Account update failed.");

}

}

} catch (SQLException e) {

e.printStackTrace();

}

}

private static boolean accountExists(Connection connection, int accountId) {

try {

String sql = "SELECT account\_id FROM accounts WHERE account\_id = " + accountId;

try (Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(sql)) {

return resultSet.next();

}

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

public static void exit() throws InterruptedException {

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

int i = 5;

while (i != 0) {

System.out.print(".");

Thread.sleep(1000);

i--;

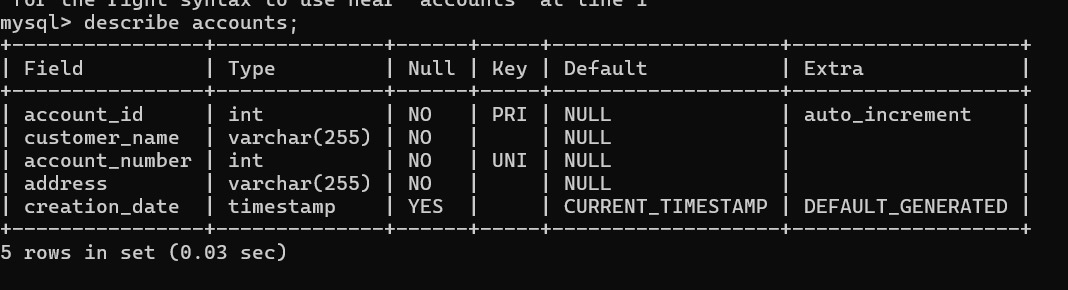
}

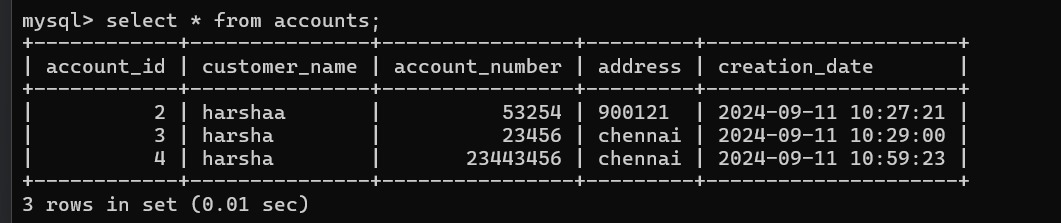
System.out.println();

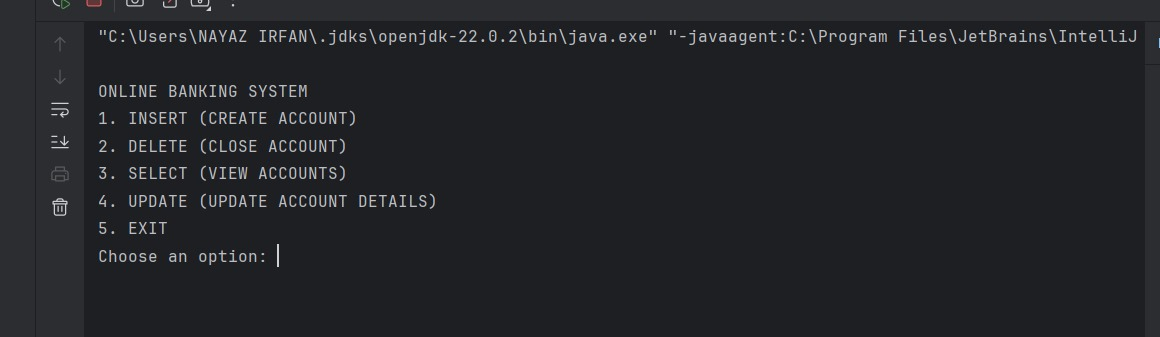
System.out.println("Thank you for using the Banking System!");

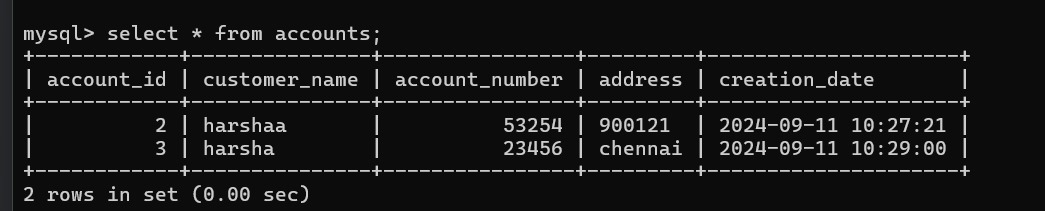
}

}





****

****

**CONCLUSION:**

In conclusion, the research underscores the importance of implementing best practices, optimizing performance, managing transactions effectively, and incorporating robust security measures in the development of online banking systems. By leveraging these insights and lessons from successful implementations, developers can create secure, efficient, and reliable banking applications that meet the high standards required for modern financial services.

**Identification of Best Practices**:

* Implementing CRUD operations effectively involves adhering to best practices that enhance the clarity, maintainability, and security of the application. Utilizing prepared statements and parameterized queries is crucial for preventing SQL injection attacks and ensuring robust data handling. Adhering to principles like the Single Responsibility Principle (SRP) promotes modular and manageable code, facilitating better maintenance and scalability.

**Performance Insights**:

* Optimization techniques such as indexing and batch processing are essential for improving the performance of online banking systems. Indexing helps in speeding up query execution, especially for frequently accessed fields such as account numbers and transaction IDs. Batch processing reduces the overhead associated with multiple database interactions, thereby improving the system’s efficiency and responsiveness. Understanding these optimization strategies allows for the development of systems that handle high volumes of transactions with minimal latency.

**Effective Transaction Management**:

* Ensuring data integrity and consistency in financial transactions is a cornerstone of any online banking system. Applying ACID properties (Atomicity, Consistency, Isolation, Durability) helps maintain a reliable and consistent state of the database, even in the face of failures or concurrent transactions. Effective transaction management strategies are necessary for handling complex scenarios, such as concurrent access and rollback operations, ensuring that the system remains dependable and accurate.

**Robust Security Measures**:

* Protecting sensitive data and preventing security breaches are paramount in online banking systems. Implementing strong security practices, including encryption, secure authentication, and regular vulnerability assessments, helps safeguard against common threats. Effective security measures are vital for maintaining user trust and protecting financial information from unauthorized access and malicious attacks.

**Real-World Insights**:

* Analysis of successful implementations of online banking systems provides practical lessons that can be applied to enhance future development efforts. Real-world case studies reveal effective strategies for scaling applications, managing data efficiently, and addressing common challenges. By learning from these examples, developers can design systems that are not only functional but also scalable and resilient to real-world demands.

**REFERENCES:**

1. **"Modern Database Management" by Jeffrey A. Hoffer, Ramesh Venkataraman, and Heikki Topi**
2. **"Database System Concepts" by Abraham Silberschatz, Henry Korth, and S. Sudarshan**
3. **"Java Persistence with Hibernate" by Christian Bauer and Gavin King**
4. **"Security Engineering: A Guide to Building Dependable Distributed Systems" by Ross J. Anderson**
5. **"Transaction Processing: Concepts and Techniques" by Jim Gray and Andreas Reuter**