**JDBC Real-Time Case Study: Vehicle Insurance Claim System**

**Scenario**:  
We are designing a simple **Vehicle Insurance Claim System** where a user can file a claim for their vehicle insurance, and the system stores and manages these claims in a database. This system will leverage JDBC (Java Database Connectivity) to interact with the database to store and retrieve claim information.

**Database Design:**

We will use a relational database with the following structure:

**Tables:**

1. **vehicles** (Stores information about vehicles insured by the company):
   * vehicle\_id (Primary Key)
   * owner\_name
   * vehicle\_model
   * license\_plate
   * insurance\_policy\_number
2. **claims** (Stores the claim details):
   * claim\_id (Primary Key)
   * vehicle\_id (Foreign Key referencing vehicles)
   * claim\_date
   * claim\_amount
   * claim\_status (Pending, Approved, Denied)

**Example Database Schema:**

sql

Copy code

CREATE TABLE vehicles (

vehicle\_id INT AUTO\_INCREMENT PRIMARY KEY,

owner\_name VARCHAR(100),

vehicle\_model VARCHAR(100),

license\_plate VARCHAR(50),

insurance\_policy\_number VARCHAR(50)

);

CREATE TABLE claims (

claim\_id INT AUTO\_INCREMENT PRIMARY KEY,

vehicle\_id INT,

claim\_date DATE,

claim\_amount DECIMAL(10, 2),

claim\_status VARCHAR(20),

FOREIGN KEY (vehicle\_id) REFERENCES vehicles(vehicle\_id)

);

**JDBC Integration:**

**1. Database Connection:**

Establish a connection to the MySQL database using JDBC.

java

Copy code

import java.sql.\*;

public class DatabaseConnection {

public static Connection getConnection() throws SQLException {

String url = "jdbc:mysql://localhost:3306/insurance\_db";

String username = "root";

String password = "password";

return DriverManager.getConnection(url, username, password);

}

}

**2. Inserting a Vehicle Record:**

Insert a new vehicle into the vehicles table.

java

Copy code

import java.sql.\*;

public class VehicleInsurance {

public static void addVehicle(String ownerName, String vehicleModel, String licensePlate, String policyNumber) {

String query = "INSERT INTO vehicles (owner\_name, vehicle\_model, license\_plate, insurance\_policy\_number) VALUES (?, ?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, ownerName);

stmt.setString(2, vehicleModel);

stmt.setString(3, licensePlate);

stmt.setString(4, policyNumber);

int rowsAffected = stmt.executeUpdate();

if (rowsAffected > 0) {

System.out.println("Vehicle record added successfully.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**3. Filing a Claim:**

Insert a new claim into the claims table for a specific vehicle.

java

Copy code

import java.sql.\*;

public class InsuranceClaim {

public static void fileClaim(int vehicleId, double claimAmount) {

String query = "INSERT INTO claims (vehicle\_id, claim\_date, claim\_amount, claim\_status) VALUES (?, CURDATE(), ?, 'Pending')";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, vehicleId); // Vehicle ID from the vehicles table

stmt.setDouble(2, claimAmount); // Claim amount

int rowsAffected = stmt.executeUpdate();

if (rowsAffected > 0) {

System.out.println("Claim filed successfully.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**4. Retrieving Claim Status:**

Retrieve the claim status for a particular vehicle.

java

Copy code

import java.sql.\*;

public class ClaimStatus {

public static void getClaimStatus(int claimId) {

String query = "SELECT claim\_status FROM claims WHERE claim\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, claimId);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

String status = rs.getString("claim\_status");

System.out.println("Claim Status: " + status);

} else {

System.out.println("Claim not found.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**5. Updating Claim Status:**

Update the status of a claim (e.g., approved or denied).

java

Copy code

import java.sql.\*;

public class ClaimApproval {

public static void updateClaimStatus(int claimId, String newStatus) {

String query = "UPDATE claims SET claim\_status = ? WHERE claim\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, newStatus); // New status (Approved/Denied)

stmt.setInt(2, claimId); // Claim ID

int rowsAffected = stmt.executeUpdate();

if (rowsAffected > 0) {

System.out.println("Claim status updated successfully.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**Real-Time Use Case Flow:**

1. **Adding a Vehicle**:  
   A user enters their vehicle information, which is then stored in the vehicles table. For example:

java

Copy code

VehicleInsurance.addVehicle("John Doe", "Toyota Camry", "ABC123", "POLICY001");

1. **Filing a Claim**:  
   A claim is filed for an insured vehicle by specifying the vehicle's ID and the claim amount. The system stores this information in the claims table.

java

Copy code

InsuranceClaim.fileClaim(1, 1500.00); // Vehicle ID 1

1. **Viewing Claim Status**:  
   The user can check the status of their claim using the claim ID.

java

Copy code

ClaimStatus.getClaimStatus(1); // Claim ID 1

1. **Approving or Denying the Claim**:  
   Based on the inspection and evaluation, the claim can be updated to "Approved" or "Denied."

java

Copy code

ClaimApproval.updateClaimStatus(1, "Approved");

**Summary:**

* The system handles insurance claims using **JDBC** for database interactions.
* The **vehicle** and **claim** tables store data related to vehicles and their claims.
* Using **PreparedStatement** ensures efficient and secure database operations.
* The claim system allows users to add vehicles, file claims, view statuses, and update claim outcomes.

This basic example demonstrates how a real-world system can interact with a relational database to manage vehicle insurance claims using JDBC.

**Executing stored procedures using CallableStatement** is a common way to interact with the database to perform complex operations like inserting records, updating data, or executing multiple queries within a single stored procedure.

Let's expand on the **Vehicle Insurance Claim System** and include the use of **stored procedures** for actions like filing a claim and updating the claim status.

**Step-by-Step Guide**

**1. Database Setup (Stored Procedure)**

First, let's create a stored procedure in the database to handle the insertion of a claim into the claims table.

**Stored Procedure:**

sql

Copy code

DELIMITER $$

CREATE PROCEDURE fileClaimProcedure(

IN vehicle\_id INT,

IN claim\_amount DECIMAL(10, 2)

)

BEGIN

INSERT INTO claims (vehicle\_id, claim\_date, claim\_amount, claim\_status)

VALUES (vehicle\_id, CURDATE(), claim\_amount, 'Pending');

END $$

DELIMITER ;

**Explanation**:

* The fileClaimProcedure procedure takes the vehicle\_id and claim\_amount as input parameters and inserts a new record into the claims table with a default claim\_status of 'Pending'.

**2. Java Code for Executing Stored Procedures**

Now that the stored procedure is ready, we will use the CallableStatement to execute the stored procedure from Java.

**Example Code:**

java

Copy code

import java.sql.\*;

public class InsuranceClaimWithStoredProcedure {

public static void fileClaim(int vehicleId, double claimAmount) {

// SQL call for the stored procedure

String sql = "{CALL fileClaimProcedure(?, ?)}";

try (Connection conn = DatabaseConnection.getConnection();

CallableStatement stmt = conn.prepareCall(sql)) {

// Set input parameters for the stored procedure

stmt.setInt(1, vehicleId); // vehicle\_id

stmt.setDouble(2, claimAmount); // claim\_amount

// Execute the stored procedure

stmt.execute();

System.out.println("Claim filed successfully.");

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**3. Calling the Stored Procedure**

To file a claim for a vehicle (say, vehicle with vehicle\_id = 1 and claim amount 2000.00), you can call the method fileClaim as follows:

java

Copy code

public class Main {

public static void main(String[] args) {

// Filing a claim for a vehicle with vehicle\_id = 1 and claim amount 2000.00

InsuranceClaimWithStoredProcedure.fileClaim(1, 2000.00);

}

}

**Explanation of the Java Code:**

* **CallableStatement**: The CallableStatement is used to execute the stored procedure. The ? placeholders in the SQL statement are replaced with the actual input parameters (vehicle\_id and claim\_amount).
* **setInt()** and **setDouble()**: These methods are used to set the input parameters for the stored procedure.
* **execute()**: This method executes the stored procedure.

**4. Handling Output Parameters (Optional)**

If your stored procedure returns a value (e.g., a result or status), you can use **output parameters**.

For example, let’s modify the stored procedure to return a status message.

**Modified Stored Procedure with Output Parameter:**

sql

Copy code

DELIMITER $$

CREATE PROCEDURE fileClaimProcedure(

IN vehicle\_id INT,

IN claim\_amount DECIMAL(10, 2),

OUT claim\_status\_message VARCHAR(255)

)

BEGIN

INSERT INTO claims (vehicle\_id, claim\_date, claim\_amount, claim\_status)

VALUES (vehicle\_id, CURDATE(), claim\_amount, 'Pending');

SET claim\_status\_message = 'Claim filed successfully.';

END $$

DELIMITER ;

**Java Code for Output Parameter:**

java

Copy code

import java.sql.\*;

public class InsuranceClaimWithOutput {

public static void fileClaim(int vehicleId, double claimAmount) {

// SQL call for the stored procedure

String sql = "{CALL fileClaimProcedure(?, ?, ?)}";

try (Connection conn = DatabaseConnection.getConnection();

CallableStatement stmt = conn.prepareCall(sql)) {

// Set input parameters for the stored procedure

stmt.setInt(1, vehicleId); // vehicle\_id

stmt.setDouble(2, claimAmount); // claim\_amount

// Register output parameter

stmt.registerOutParameter(3, Types.VARCHAR); // claim\_status\_message

// Execute the stored procedure

stmt.execute();

// Retrieve the output parameter

String statusMessage = stmt.getString(3);

System.out.println(statusMessage); // Print the message returned by the procedure

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**5. Conclusion**

* **Stored Procedures** help encapsulate business logic at the database level, improving performance, maintainability, and security.
* Using **CallableStatement** in Java, you can efficiently interact with stored procedures for operations like inserting claims.
* The provided example shows how to file a claim using a stored procedure and how to handle output parameters for getting success messages from the procedure.

This approach makes your application more modular, as the database handles the logic of the procedure, while Java only interacts with it through JDBC.