To successfully complete this week assignment, we will need to ensure that our Java application can interact with the MySQL database. Begin by setting the appropriate permissions for the JavaBean to access the csd430 database. This involves configuring your MySQL user privileges to allow SELECT operations on the steve\_movies\_data table.

In our JSP page, utilize scriptlets to fetch the data and populate an HTML dropdown menu with the unique keys from the database. When a user selects a key and submits the form, the corresponding record should be retrieved and displayed in a well-structured HTML table.

**Key Concepts**

Before diving into the code, let's clarify some key concepts:

* **JDBC**: A Java API that allows Java applications to interact with databases.
* **Connection**: Represents a connection to a specific database.
* **Statement**: Used to execute SQL queries against the database.
* **SQLException**: An exception that provides information on a database access error or other errors.

**Code Structure**

The code is structured into a single class named CreateDatabase.java, which contains the main method. The main method is responsible for establishing a connection to the database, executing SQL commands to create a database and a table, and inserting data into that table.

*Breakdown of the Code:*

1. **Database Credentials**: The code defines constants for the database URL, username, and password.
2. **Connection and Statement Initialization**: It initializes the Connection and Statement objects.
3. **Database Operations**: The program performs several operations:
   * Drops the existing database if it exists.
   * Creates a new database.
   * Creates a table within that database.
   * Inserts multiple records into the table.
4. **Error Handling**: The code includes try-catch blocks to handle SQL and general exceptions.
5. **Resource Cleanup**: Finally, it ensures that resources are closed properly.

## CreateDatabase.java

**package** com.example;

// Steve Stylin Module 5 and 6 part 1: Project

**import** java.sql.Connection; // Importing the Connection class for database connection

**import** java.sql.DriverManager; // Importing DriverManager to manage database connections

**import** java.sql.Statement; // Importing Statement for executing SQL statements

**import** java.sql.SQLException; // Importing SQLException for handling SQL exceptions

**public** **class** CreateDatabase {

// Database credentials

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost:3306/"; // URL for the database connection

**static** **final** String ***USER*** = "student1"; // Database username

**static** **final** String ***PASS*** = "pass"; // Database password

**public** **static** **void** main(String[] args) {

Connection conn = **null**; // Initialize connection variable

Statement stmt = **null**; // Initialize statement variable

**try** {

// Register JDBC driver

Class.*forName*("com.mysql.cj.jdbc.Driver"); // Load MySQL JDBC driver

// Open a connection

conn = DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***); // Establish connection to the database

System.***out***.println("Connection established successfully."); // Confirm successful connection

// Create statement

stmt = conn.createStatement(); // Create a statement object for executing SQL queries

// Drop the database if it exists

String sql = "DROP DATABASE IF EXISTS csd430"; // SQL command to drop the database if it exists

stmt.executeUpdate(sql); // Execute the SQL command

// Create database

sql = "CREATE DATABASE csd430"; // SQL command to create a new database

stmt.executeUpdate(sql); // Execute the SQL command

System.***out***.println("Database 'csd430' created successfully."); // Confirm database creation

// Use the new database

stmt.executeUpdate("USE csd430"); // Set the context to the newly created database

// Create table

sql = "CREATE TABLE steve\_movies\_data (" + // SQL command to create a new table

"id INT NOT NULL AUTO\_INCREMENT, " + // Define 'id' column as an auto-incrementing integer

"title VARCHAR(255) NOT NULL, " + // Define 'title' column as a non-null string

"genres VARCHAR(255), " + // Define 'genres' column as a string

"release\_date DATE, " + // Define 'release\_date' column as a date

"imdb\_rating DECIMAL(3,1), " + // Define 'imdb\_rating' column as a decimal

"PRIMARY KEY (id))"; // Set 'id' as the primary key

stmt.executeUpdate(sql); // Execute the SQL command

System.***out***.println("Table 'steve\_movies\_data' created successfully."); // Confirm table creation

// Insert values into the table

String[] movieValues = { // Array of movie data to be inserted

"('300: Rise of an Empire', 'Action, Drama', '2014-03-07', 6.2)",

"('Non-Stop', 'Action, Mystery, Thriller', '2014-02-28', 6.9)",

"('Son of God', 'Biography, Drama, History', '2014-02-28', 5.7)",

"('Noah', 'Action, Adventure, Drama', '2014-03-28', 5.8)",

"('12 Years a Slave', 'Biography, Drama, History', '2013-11-08', 8.1)",

"('The Passion of the Christ', 'Drama', '2004-02-25', 7.3)",

"('Need for Speed', 'Action, Crime, Drama, Thriller', '2014-03-14', 6.4)",

"('Lone Survivor', 'Action, Biography, Drama, War', '2014-01-10', 7.5)",

"('RoboCop', 'Action, Crime, Sci-Fi, Thriller', '2014-02-12', 6.1)",

"('Captain America: The Winter Soldier', 'Action, Adventure, Sci-Fi, Thriller', '2014-04-04', 7.7)"

};

**for** (String value : movieValues) { // Loop through each movie value

sql = "INSERT INTO steve\_movies\_data (title, genres, release\_date, imdb\_rating) VALUES " + value; // SQL command to insert movie data

stmt.executeUpdate(sql); // Execute the SQL command

}

System.***out***.println("Table 'steve\_movies\_data' populated successfully!"); // Confirm table population

} **catch** (SQLException se) { // Handle SQL exceptions

se.printStackTrace(); // Print stack trace for debugging

} **catch** (Exception e) { // Handle general exceptions

e.printStackTrace(); // Print stack trace for debugging

} **finally** {

// Clean up environment

**try** {

**if** (stmt != **null**) stmt.close(); // Close statement if not null

**if** (conn != **null**) conn.close(); // Close connection if not null

System.***out***.println("Connection closed."); // Confirm connection closure

} **catch** (SQLException se) { // Handle SQL exceptions during cleanup

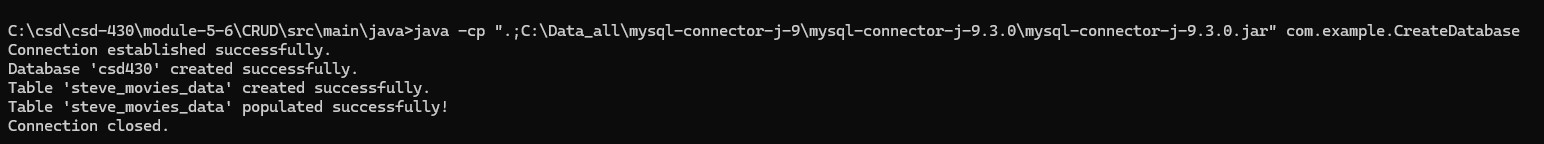
se.printStackTrace(); // Print stack trace for debugging

}

}

}

Output of the java file:



The structure of the Movie class is straightforward and follows Java conventions:

1. **Encapsulation**: The class encapsulates the movie's attributes (ID, title, genres, release date, and IMDb rating) as private variables, ensuring that they are not directly accessible from outside the class.
2. **Constructors**: It includes both a no-argument constructor and a parameterized constructor, allowing for flexibility in object creation.
3. **Getters and Setters**: The class provides public methods to access and modify the private variables, adhering to the principles of encapsulation.

## Movie.java

**package** com.example; // Package declaration for organizing classes

//Steve Stylin Module 5 and 6 part 1: Project

**import** java.util.Date; // Importing the Date class from java.util package

**public** **class** Movie { // Declaration of the Movie class

**private** **int** id; // Private variable to store the movie ID

**private** String title; // Private variable to store the movie title

**private** String genres; // Private variable to store the movie genres

**private** Date releaseDate; // Private variable to store the release date of the movie

**private** **double** imdbRating; // Private variable to store the IMDb rating of the movie

// No-argument constructor

**public** Movie() {

}

// Parameterized constructor to initialize Movie object with specific values

**public** Movie(**int** id, String title, String genres, Date releaseDate, **double** imdbRating) {

**this**.id = id; // Assigning the provided id to the class variable

**this**.title = title; // Assigning the provided title to the class variable

**this**.genres = genres; // Assigning the provided genres to the class variable

**this**.releaseDate = releaseDate; // Assigning the provided release date to the class variable

**this**.imdbRating = imdbRating; // Assigning the provided IMDb rating to the class variable

}

// Getter method for id

**public** **int** getId() { **return** id; } // Returns the movie ID

// Setter method for id

**public** **void** setId(**int** id) { **this**.id = id; } // Sets the movie ID

// Getter method for title

**public** String getTitle() { **return** title; } // Returns the movie title

// Setter method for title

**public** **void** setTitle(String title) { **this**.title = title; } // Sets the movie title

// Getter method for genres

**public** String getGenres() { **return** genres; } // Returns the movie genres

// Setter method for genres

**public** **void** setGenres(String genres) { **this**.genres = genres; } // Sets the movie genres

// Getter method for releaseDate

**public** Date getReleaseDate() { **return** releaseDate; } // Returns the release date of the movie

// Setter method for releaseDate

**public** **void** setReleaseDate(Date releaseDate) { **this**.releaseDate = releaseDate; } // Sets the release date of the movie

// Getter method for imdbRating

**public** **double** getImdbRating() { **return** imdbRating; } // Returns the IMDb rating of the movie

// Setter method for imdbRating

**public** **void** setImdbRating(**double** imdbRating) { **this**.imdbRating = imdbRating; } // Sets the IMDb rating of the movie

}

The MovieBean class is a Java application designed to interact with a MySQL database, specifically to manage movie data. It establishes a connection to the database, retrieves a list of movies, and allows for fetching individual movie details based on their ID. This class serves as a foundational component for applications that require managing movie data.

1. **JDBC (Java Database Connectivity)**: This is the API that allows Java applications to interact with databases. The MovieBean class utilizes JDBC to connect to a MySQL database.
2. **Prepared Statements**: These are used to execute SQL queries safely and efficiently, preventing SQL injection attacks.
3. **ResultSet**: This is a Java object that holds the data retrieved from a database after executing a query.
4. **Object-Oriented Programming**: The class encapsulates data and methods related to movie management, promoting reusability and modularity.

**package** com.example; // Package declaration for the MovieBean class

//Steve Stylin Module 5 and 6 part 1: Project

**import** java.sql.\*; // Importing SQL classes for database operations

**import** java.util.ArrayList; // Importing ArrayList for dynamic array functionality

**import** java.util.List; // Importing List interface for movie list

**public** **class** MovieBean {

**private** String jdbcURL = "jdbc:mysql://localhost:3306/csd430"; // JDBC URL for MySQL database connection

**private** String jdbcUsername = "student1"; // Database username

**private** String jdbcPassword = "pass"; // Database password

// Method to test the database connection

**public** **void** testConnection() {

**try** (Connection connection = DriverManager.*getConnection*(jdbcURL, jdbcUsername, jdbcPassword)) { // Establishing connection

**if** (connection != **null**) { // Checking if the connection is successful

System.***out***.println("Connection successful!"); // Output success message

} **else** {

System.***out***.println("Failed to make connection!"); // Output failure message

}

} **catch** (SQLException e) { // Catching SQL exceptions

e.printStackTrace(); // Print stack trace for debugging

}

}

// Method to retrieve all movies from the database

**public** List<Movie> getAllMovies() {

List<Movie> movies = **new** ArrayList<>(); // Initializing a list to hold Movie objects

String sql = "SELECT \* FROM steve\_movies\_data"; // SQL query to select all movies

**try** (Connection connection = DriverManager.*getConnection*(jdbcURL, jdbcUsername, jdbcPassword); // Establishing connection

PreparedStatement statement = connection.prepareStatement(sql); // Preparing SQL statement

ResultSet resultSet = statement.executeQuery()) { // Executing the query and getting results

**while** (resultSet.next()) { // Iterating through the result set

**int** id = resultSet.getInt("id"); // Retrieving movie ID

String title = resultSet.getString("title"); // Retrieving movie title

String genres = resultSet.getString("genres"); // Retrieving movie genres

Date releaseDate = resultSet.getDate("release\_date"); // Retrieving release date

**double** imdbRating = resultSet.getDouble("imdb\_rating"); // Retrieving IMDb rating

movies.add(**new** Movie(id, title, genres, releaseDate, imdbRating)); // Adding movie to the list

}

} **catch** (SQLException e) { // Catching SQL exceptions

e.printStackTrace(); // Print stack trace for debugging

}

**return** movies; // Returning the list of movies

}

// Method to retrieve a movie by its ID

**public** Movie getMovieById(**int** id) {

Movie movie = **null**; // Initializing movie variable

String sql = "SELECT \* FROM steve\_movies\_data WHERE id = ?"; // SQL query with placeholder for ID

**try** (Connection connection = DriverManager.*getConnection*(jdbcURL, jdbcUsername, jdbcPassword); // Establishing connection

PreparedStatement statement = connection.prepareStatement(sql)) { // Preparing SQL statement

statement.setInt(1, id); // Setting the ID parameter in the SQL query

ResultSet resultSet = statement.executeQuery(); // Executing the query

**if** (resultSet.next()) { // Checking if a result is returned

String title = resultSet.getString("title"); // Retrieving movie title

String genres = resultSet.getString("genres"); // Retrieving movie genres

Date releaseDate = resultSet.getDate("release\_date"); // Retrieving release date

**double** imdbRating = resultSet.getDouble("imdb\_rating"); // Retrieving IMDb rating

movie = **new** Movie(id, title, genres, releaseDate, imdbRating); // Creating movie object

}

} **catch** (SQLException e) { // Catching SQL exceptions

e.printStackTrace(); // Print stack trace for debugging

}

**return** movie; // Returning the movie object

}

// Main method to run the application

**public** **static** **void** main(String[] args) {

MovieBean movieBean = **new** MovieBean(); // Creating an instance of MovieBean

movieBean.testConnection(); // Call the method to test the connection

}

}

Output: 

1. **JavaServer Pages (JSP)**: JSP is a technology that helps create dynamically generated web pages based on HTML, XML, or other document types. It allows embedding Java code directly into the HTML.
2. **HTML Structure**: The code utilizes standard HTML elements to create a structured layout, including headings, lists, and links.
3. **Styling**: Basic CSS is applied to enhance the visual appearance of the page, specifically setting a background color.
4. **Dynamic Content**: The use of Java imports suggests that the page may interact with Java classes, such as MovieBean and Movie, to manage movie-related data.

The index.jsp is the welcome page of the application, and you can navigate back and forth through the program.

Index.jsp

<%@ page import=*"java.util.List"* %> <!-- Importing the List class from the java.util package -->

<%@ page import=*"com.example.MovieBean"* %> <!-- Importing the MovieBean class from the com.example package -->

<%@ page import=*"com.example.Movie"* %> <!-- Importing the Movie class from the com.example package -->

<%@ page contentType=*"text/html;charset=UTF-8"* language=*"java"* %> <!-- Setting the content type to HTML with UTF-8 encoding -->

<!DOCTYPE html>

<html>

<head>

<title>Document and Movie Selection</title>

<style>

**body** {

background-color: *#D3D3D3*; <!-- Setting the background color of the body to light gray -->

}

</style>

</head>

<body>

<h1>Word Document List</h1>

<ul>

<!-- Replace the items below with the actual paths to your Word documents -->

<li><a href=*"documents/Stylin\_\_movies\_data.docx"*>Module 5.1 Documentation</a></li> <!-- Link to Module 5.1 documentation -->

<li><a href=*"documents/Stylin\_module6\_databse\_connection.docx"*>Module 6.1 Part 1</a></li> <!-- Link to Module 6.1 Part 1 documentation -->

</ul>

<h2>Select a Movie</h2>

<p><a href=*"selectMovie.jsp"*>Click here to select a movie</a></p> <!-- Link to the movie selection page -->

<footer>

<p>Author: Steve Stylin</p> <!-- Displaying the author's name -->

<p>Modules 5 and 6 Assignment - Project Part 1</p> <!-- Description of the assignment -->

<p>School: Bellevue University</p> <!-- Displaying the school name -->

<p>Professor: Darrell Payne</p> <!-- Displaying the professor's name -->

</footer>

</body>

</html>

A screenshot of a computer

AI-generated content may be incorrect.

The selectMovie.jsp

Allow user to select a movie, from a populate list of exiting movies on the database.

<%@ page import=*"java.sql.\*"* %>

<%@ page contentType=*"text/html;charset=UTF-8"* language=*"java"* %>

<html>

<head>

<title>Select a Movie</title>

</head>

<body>

<h1>Select Movie</h1>

<form action=*"displayMovie.jsp"* method=*"post"*>

<label for=*"movieId"*>Select Movie ID:</label>

<select name=*"movieId"* id=*"movieId"*>

<%

// Database connection parameters

String url = "jdbc:mysql://localhost:3306/csd430";

String user = "student1";

String password = "pass";

Connection conn = **null**;

Statement stmt = **null**;

ResultSet rs = **null**;

**try** {

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

conn = DriverManager.getConnection(url, user, password);

stmt = conn.createStatement();

String sql = "SELECT id FROM steve\_movies\_data";

rs = stmt.executeQuery(sql);

**if** (!rs.isBeforeFirst()) {

out.println("<option value=''>No movies available</option>");

} **else** {

**while** (rs.next()) {

**int** movieId = rs.getInt("id");

%>

<option value=*"*<%= movieId %>*"*><%= movieId %></option>

<%

}

}

} **catch** (Exception e) {

out.println("<option value=''>Error loading movies</option>");

// Print stack trace to HTML for debugging (development only)

java.io.StringWriter sw = **new** java.io.StringWriter();

java.io.PrintWriter pw = **new** java.io.PrintWriter(sw);

e.printStackTrace(pw);

out.println("<pre>" + sw.toString() + "</pre>");

} **finally** {

**if** (rs != **null**) **try** { rs.close(); } **catch** (SQLException ignore) {}

**if** (stmt != **null**) **try** { stmt.close(); } **catch** (SQLException ignore) {}

**if** (conn != **null**) **try** { conn.close(); } **catch** (SQLException ignore) {}

}

%>

</select>

<br><br>

<input type=*"submit"* value=*"Display Movie"*>

</form>

</body>

</html>

A close-up of a sign

AI-generated content may be incorrect.

The displayMovie.jsp is designed to display details of a movie based on the movie ID passed as a request parameter. It begins by importing necessary classes, specifically MovieBean and Movie, which are likely used for data handling. The selected movie is displayed in a nicely formatted table for reading accuracy.

<%@ page import=*"com.example.MovieBean"* %>

<%@ page import=*"com.example.Movie"* %>

<%@ page contentType=*"text/html;charset=UTF-8"* language=*"java"* %>

<html>

<head>

<title>Movie Details</title>

<style>

**table** {

border-collapse: *collapse*;

width: *50%*;

margin: *20px auto*;

}

**th,** **td** {

border: *1px solid black*;

text-align: *left*;

padding: *8px*;

}

**th** {

background-color: *#f2f2f2*;

}

</style>

</head>

<body>

<h1>Movie Details</h1>

<%

String movieIdParam = request.getParameter("movieId");

**if** (movieIdParam != **null** && !movieIdParam.isEmpty()) {

**try** {

**int** movieId = Integer.parseInt(movieIdParam);

MovieBean movieBean = **new** MovieBean();

Movie movie = movieBean.getMovieById(movieId);

**if** (movie != **null**) {

%>

<table>

<tr>

<th>ID</th>

<th>Title</th>

<th>Genres</th>

<th>Release Date</th>

<th>IMDB Rating</th>

</tr>

<tr>

<td><%= movie.getId() %></td>

<td><%= movie.getTitle() %></td>

<td><%= movie.getGenres() %></td>

<td><%= movie.getReleaseDate() %></td>

<td><%= movie.getImdbRating() %></td>

</tr>

</table>

<%

} **else** {

out.println("<p>No movie found with ID: " + movieId + "</p>");

}

} **catch** (NumberFormatException e) {

out.println("<p>Invalid movie ID format.</p>");

}

} **else** {

out.println("<p>No movie ID provided.</p>");

}

%>

<br>

<a href=*"selectMovie.jsp"*>Back to Selection</a>

<br>

<a href=*"index.jsp"*>Back to Welcome</a>

</body>

</html>

A computer screen shot of a computer screen

AI-generated content may be incorrect.

The project successfully integrates a JavaServer Pages (JSP) interface with a MySQL database to allow users to select a movie from a dynamically populated drop-down list. By utilizing JDBC for database connectivity, the application fetches movie IDs from the steve\_movies\_data table and displays them in the form. This setup not only provides a user-friendly way to select a movie but also demonstrates the fundamental concepts of web development, database interaction, and error handling in Java. The project highlights best practices, such as closing database resources to prevent memory leaks and displaying user-friendly messages in error cases. The modular structure allows for future enhancements, such as including more details about each movie or integrating additional functionalities, like user reviews or ratings. Overall, this project lays a strong foundation for developing interactive web applications and reinforces the importance of clean coding and error management in software development.