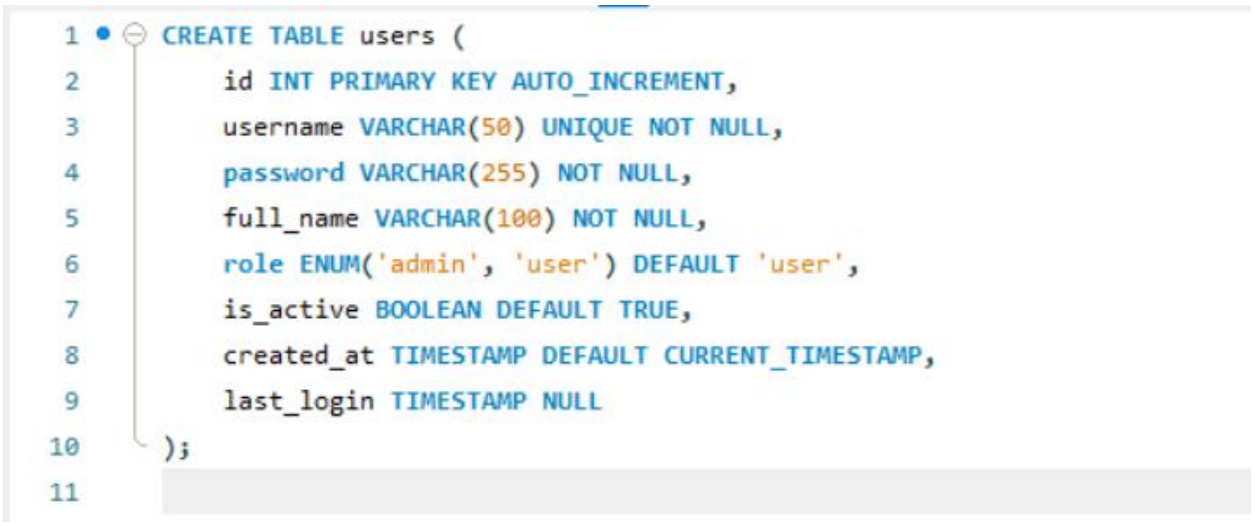
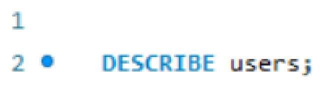
Code flow:

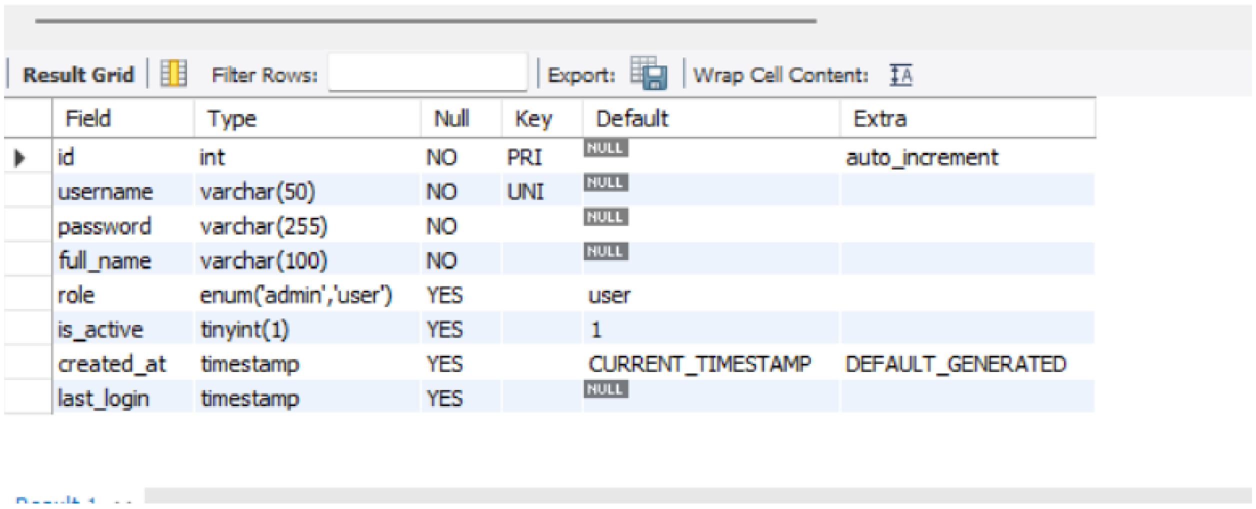
Task 1.1: Create Users Table



```sql
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    role ENUM('admin', 'user') DEFAULT 'user',
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP NULL
);
```



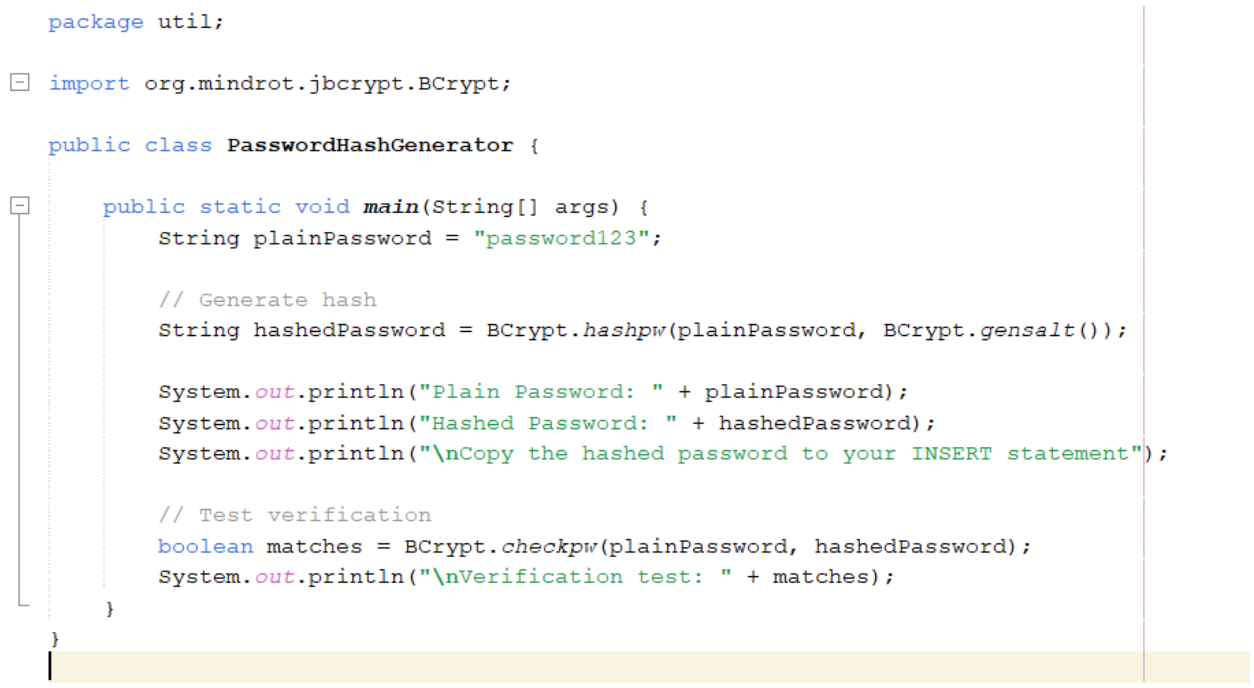
```sql

DESCRIBE users;
```



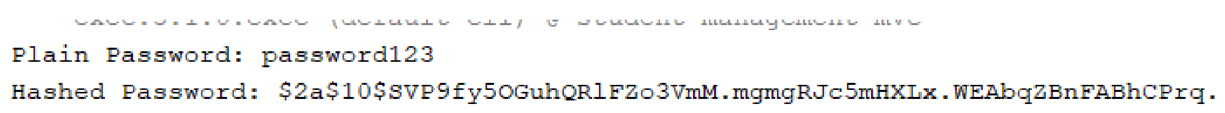
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
username	varchar(50)	NO	UNI	NULL	
password	varchar(255)	NO		NULL	
full_name	varchar(100)	NO		NULL	
role	enum('admin','user')	YES		user	
is_active	tinyint(1)	YES		1	
created_at	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
last_login	timestamp	YES		NULL	

Task 1.2: Generate Hashed Passwords

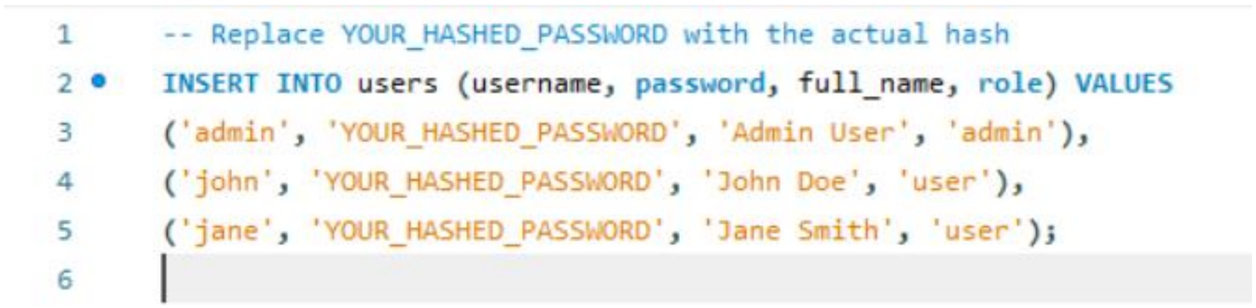


```java
package util;

import org.mindrot.jbcrypt.BCrypt;

public class PasswordHashGenerator {

    public static void main(String[] args) {
        String plainPassword = "password123";

        // Generate hash
        String hashedPassword = BCrypt.hashpw(plainPassword, BCrypt.gensalt());

        System.out.println("Plain Password: " + plainPassword);
        System.out.println("Hashed Password: " + hashedPassword);
        System.out.println("\nCopy the hashed password to your INSERT statement");

        // Test verification
        boolean matches = BCrypt.checkpw(plainPassword, hashedPassword);
        System.out.println("\nVerification test: " + matches);
    }
}
```

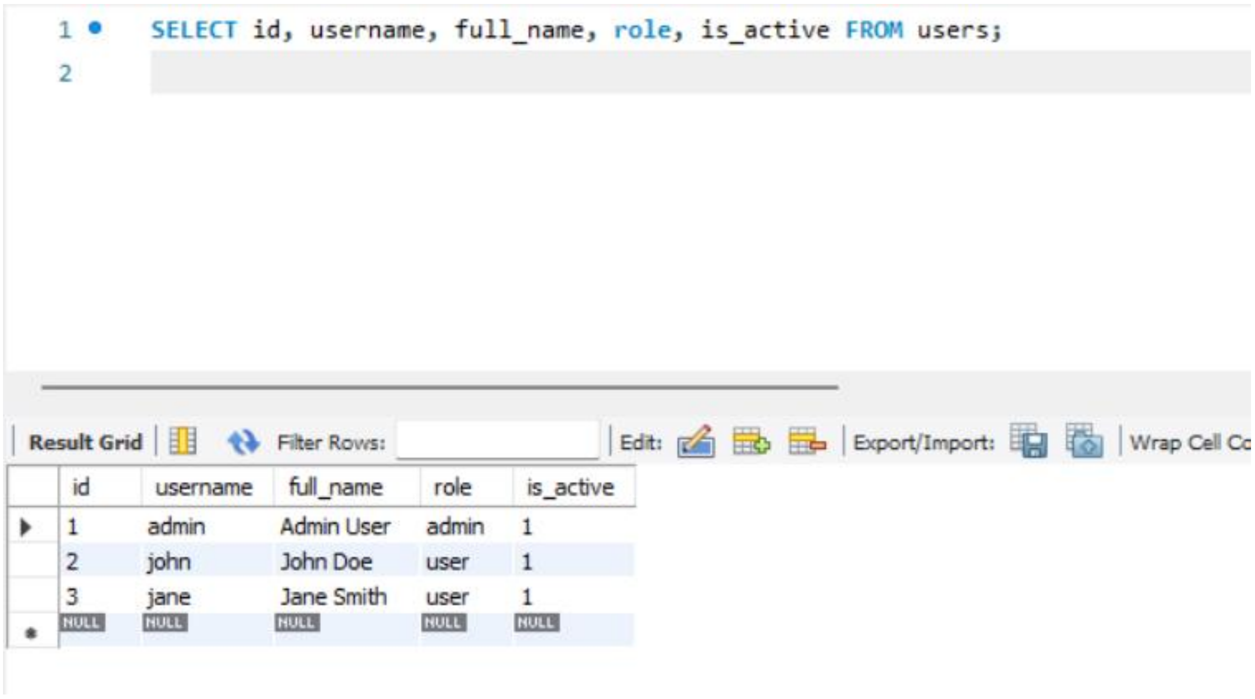


```
Plain Password: password123
Hashed Password: $2a$10$SVP9fy5OGuhQRlFZo3VmM.mgmgRJc5mHXLx.WEAbqZBnFABhCPrq.
```

Task 1.3: Insert Test Users



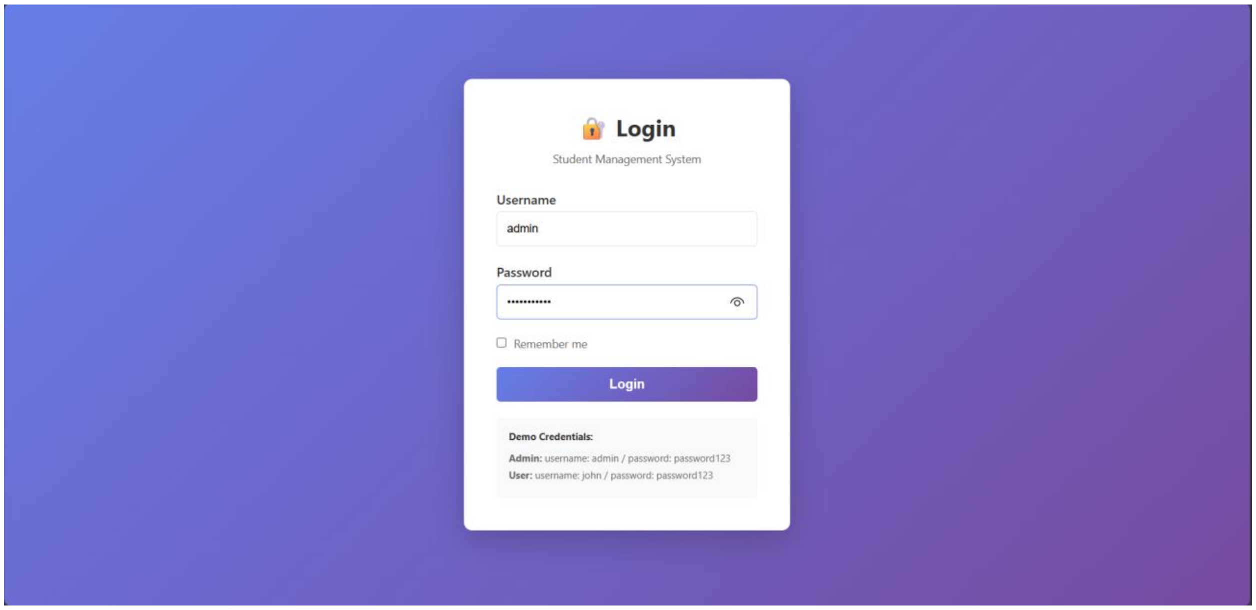
```sql
-- Replace YOUR_HASHED_PASSWORD with the actual hash
INSERT INTO users (username, password, full_name, role) VALUES
('admin', 'YOUR_HASHED_PASSWORD', 'Admin User', 'admin'),
('john', 'YOUR_HASHED_PASSWORD', 'John Doe', 'user'),
('jane', 'YOUR_HASHED_PASSWORD', 'Jane Smith', 'user');
```



```sql
SELECT id, username, full_name, role, is_active FROM users;
```

id	username	full_name	role	is_active
1	admin	Admin User	admin	1
2	john	John Doe	user	1
3	jane	Jane Smith	user	1
NULL	NULL	NULL	NULL	NULL

Task 2.1: Create User Model



🔐 **Login**

Student Management System

Username: admin

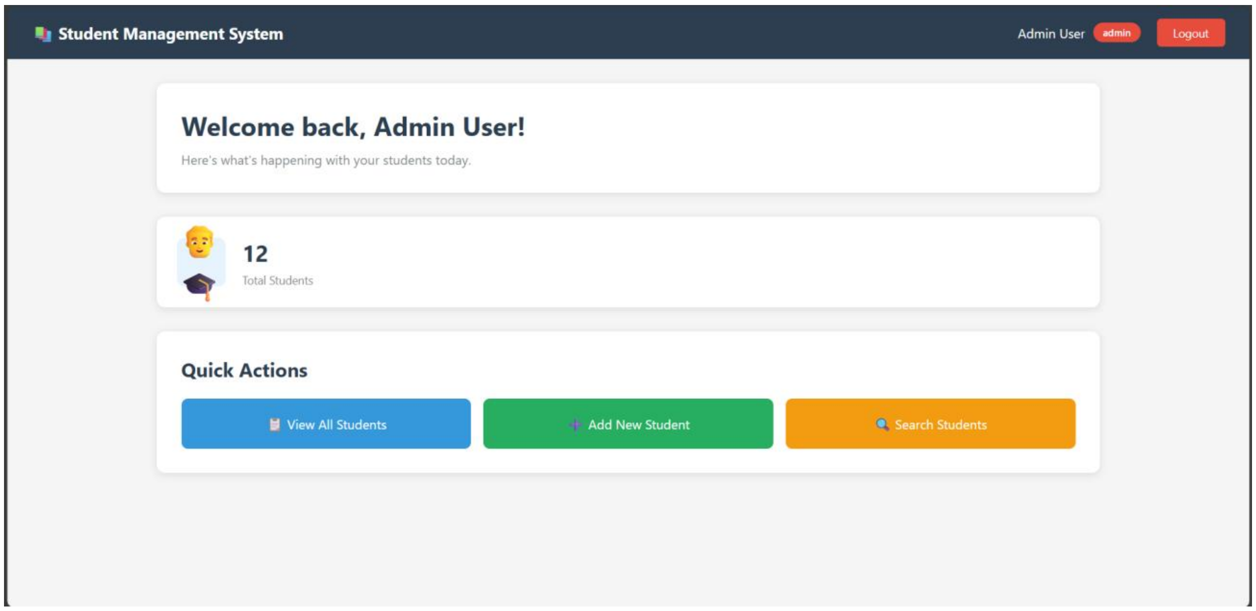
Password: ••••••••••••

☐ Remember me

Demo Credentials:

Admin: username: admin / password: password123

User: username: john / password: password123


Student Management System

Admin User admin Logout

Welcome back, Admin User!

Here's what's happening with your students today.

12

Total Students

Quick Actions

Task 2.2: Create UserDAO

1. Database Connection Method

```
// Get database connection
private Connection getConnection() throws SQLException {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
    } catch (ClassNotFoundException e) {
        throw new SQLException("MySQL Driver not found", e);
    }
}
```

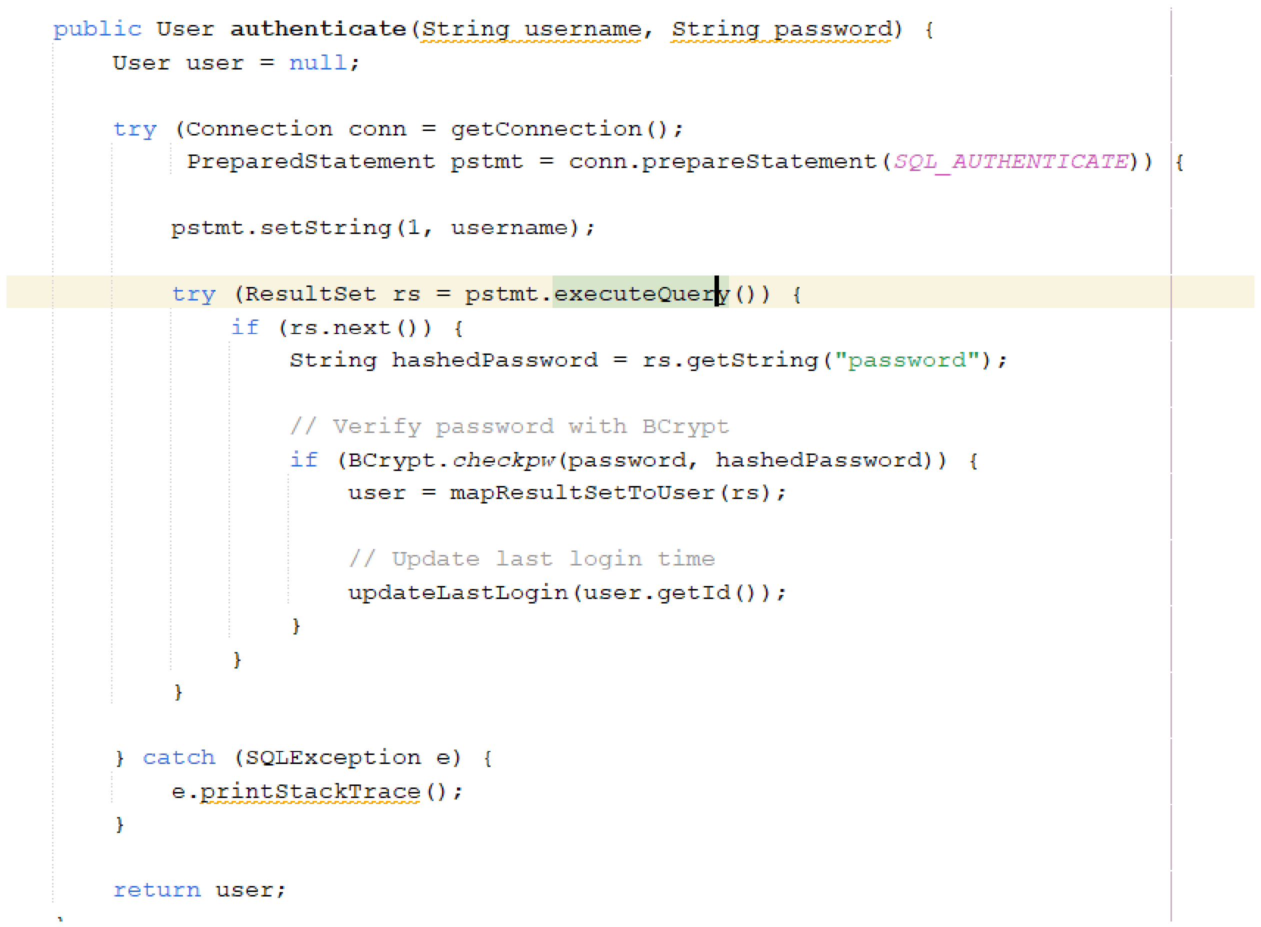
Driver Loading: It first loads the MySQL driver into memory (Class.forName("com.mysql.cj.jdbc.Driver")).

Session Creation: It uses DriverManager to open a tunnel to your database using the constants DB URL, DB USER, and DB PASSWORD.

Return: It returns a simplified Connection object that allows SQL commands to be sent.

```
private static final String DB_URL = "jdbc:mysql://localhost:3306/student_management";
private static final String DB_USER = "root";
private static final String DB_PASSWORD = "Huyhoang12a1";
```

2. Authenticate(username, password) method



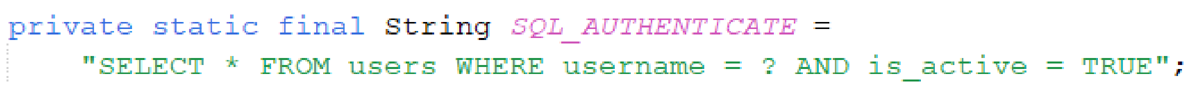
```java
public User authenticate(String username, String password) {
    User user = null;

    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(SQL_AUTHENTICATE)) {

        pstmt.setString(1, username);

        try (ResultSet rs = pstmt.executeQuery()) {
            if (rs.next()) {
                String hashedPassword = rs.getString("password");

                // Verify password with BCrypt
                if (BCrypt.checkpw(password, hashedPassword)) {
                    user = mapResultSetToUser(rs);

                    // Update last login time
                    updateLastLogin(user.getId());
                }
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return user;
```

Fetch user by username

Verify password using BCrypt.checkpw(plain, hash)

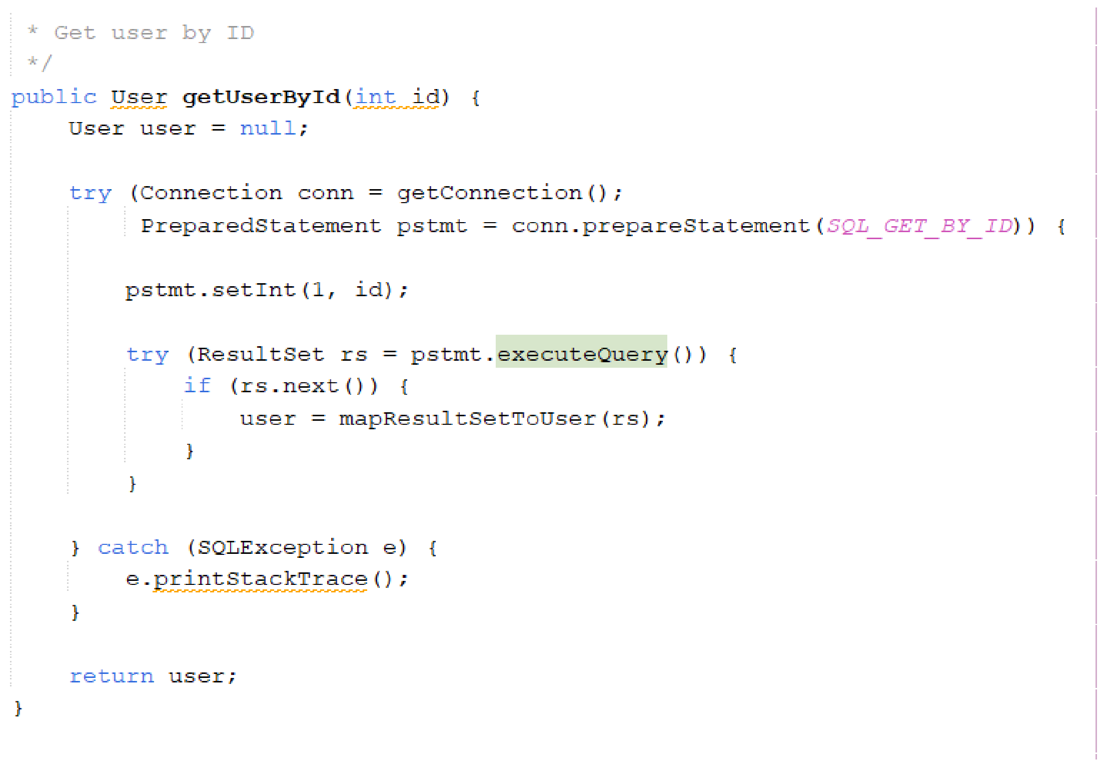
If valid → updateLastLogin(userId)

Return User object or null



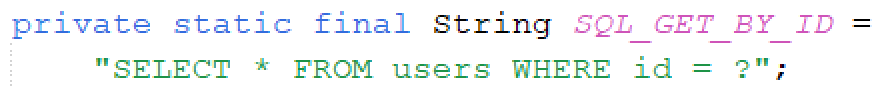
```java
private static final String SQL_AUTHENTICATE =
    "SELECT * FROM users WHERE username = ? AND is_active = TRUE";
```

3. getUserById(id) method



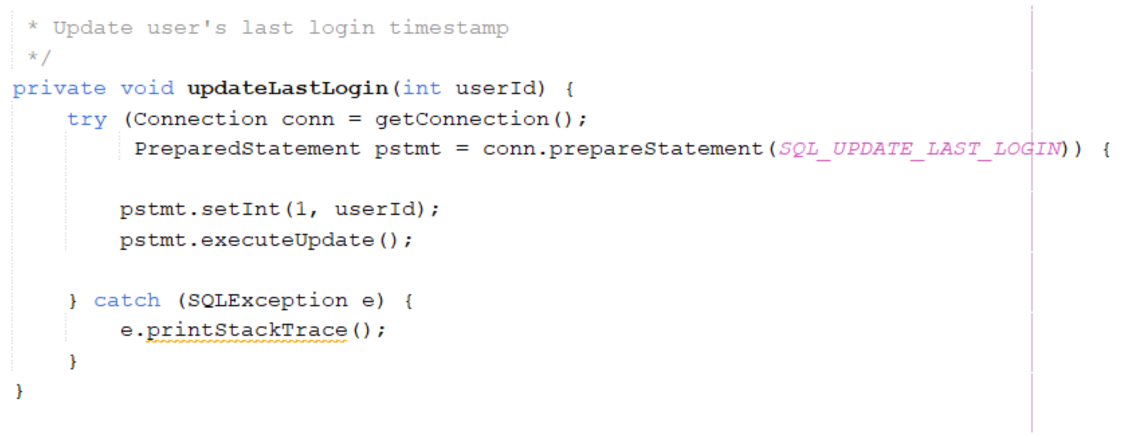
```java
 * Get user by ID
 */
public User getUserById(int id) {
    User user = null;

    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(SQL_GET_BY_ID)) {

        pstmt.setInt(1, id);

        try (ResultSet rs = pstmt.executeQuery()) {
            if (rs.next()) {
                user = mapResultSetToUser(rs);
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return user;
}
```

Retrieve a user by their ID.



```java
private static final String SQL_GET_BY_ID =
    "SELECT * FROM users WHERE id = ?";
```

4. updateLastLogin(userId) method



```java
 * Update user's last login timestamp
 */
private void updateLastLogin(int userId) {
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(SQL_UPDATE_LAST_LOGIN)) {

        pstmt.setInt(1, userId);
        pstmt.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Update the user's last login timestamp (set to NOW()).

```
private static final String SQL_UPDATE_LAST_LOGIN =
    "UPDATE users SET last_login = NOW() WHERE id = ?";
```

5. Use BCrypt for password verification

Passwords are stored hashed using

```
// Hash password before storing
String hashedPassword = BCrypt.hashpw(user.getPassword(), BCrypt.gensalt());
```

During login, compare with

```
BCrypt.checkpw(plainPassword, hashedPassword);
```

```
        // Verify password with BCrypt
        if (BCrypt.checkpw(password, hashedPassword)) {
            user = mapResultSetToUser(rs);

            // Update last login time
            updateLastLogin(user.getId());
        }
}
```