

# 实验\_数据库编程

小组成员

姓名	学号
王何佳	2023211603
叶于琳	2023211606
吴贞浩	2023211623

## 实验环境与工具：

操作系统：windows

java开发环境工具：JDK

IDE：IDEA

数据库：SQL Server

管理工具：SSMS

其他：JDBC驱动程序

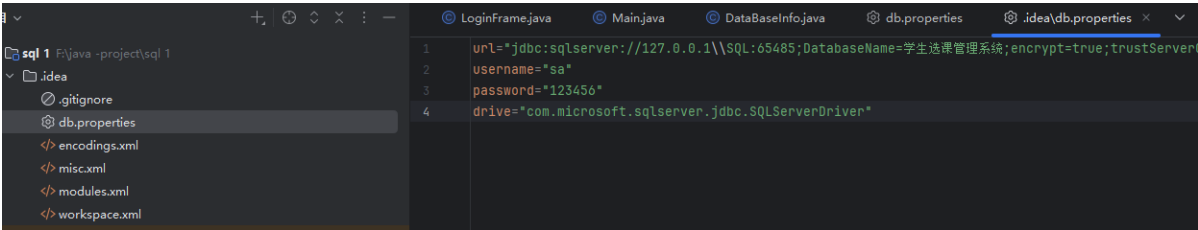
## 实验内容及步骤：

1.安装java环境和JDBC驱动程序

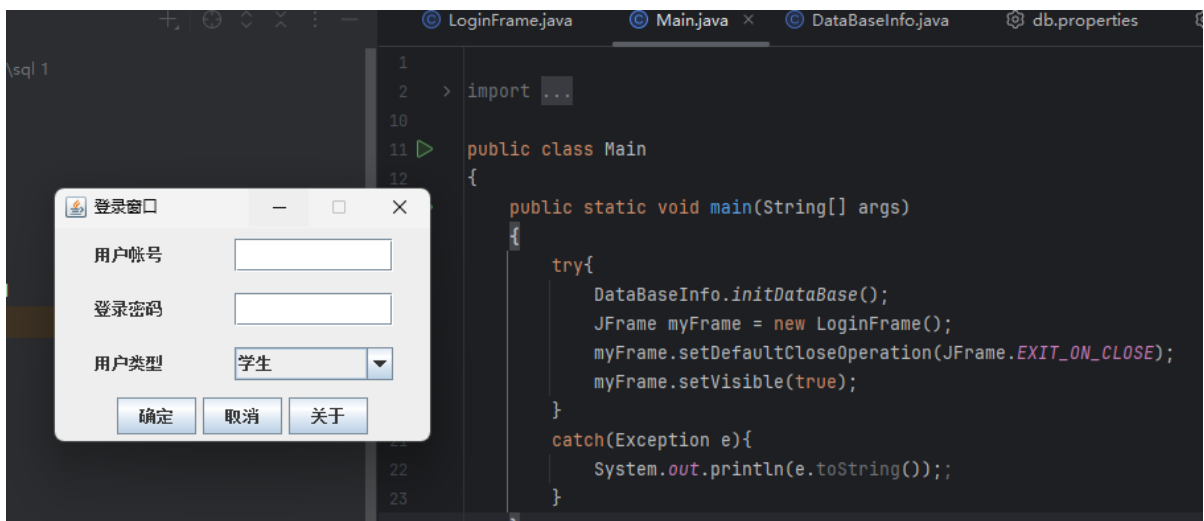
2.导入“学生选课管理系统数据库”



3.根据数据库连接信息，修改.idea和/db.properties文件，配置正确信息，使java程序可以连接到数据库



运行程序（main）



输入账号密码即可正常登录（教师账号1001，密码000）



## 4.查看LoginFrame.java源代码，理清登录过程的程序执行逻辑

### 1. 初始化与界面加载

1. **窗口创建**：LoginFrame 类继承自 JFrame，构造函数中设置窗口标题、大小和位置。
2. 界面组件初始化：
  - 创建主面板（使用 GridBagLayout 布局管理器）
  - 添加用户名、密码输入框和用户类型下拉框（学生 / 教师 / 管理员）
  - 创建按钮面板并添加 "确定"、"取消"、"关于" 三个按钮
3. 事件监听器注册：
  - 为用户类型下拉框添加选择变更监听器
  - 为三个按钮分别添加点击事件监听器

### 2. 用户交互与数据收集

1. 用户输入  
：
  - 在文本框中输入用户名和密码
  - 从下拉框选择用户类型（默认为 "学生"）

2. 点击 "确定" 按钮:

- 触发 `ActionListener` 中的 `actionPerformed` 方法
- 调用 `loginDispose()` 方法进行登录验证

### 3. 数据库验证流程

`loginDispose()` 方法的执行步骤:

1. 数据库连接:

```
Class.forName(DataBaseInfo.drive); // 加载数据库驱动
loginConnection = DriverManager.getConnection(
    DataBaseInfo.url,
    DataBaseInfo.username,
    DataBaseInfo.password
);
```

2. SQL 查询构造:

- 根据用户类型选择不同的表进行查询
- 直接拼接用户输入的用户名和密码到 SQL 语句中 (存在 SQL 注入风险)

```
if(selectedItem.equals("教师"))
    loginQuery = "SELECT * FROM 教师表 WHERE(登陆帐号='" + loginUserName + "' AND 登陆密码='" + loginPassword + "')";
else if(selectedItem.equals("管理员"))
    loginQuery = "SELECT * FROM 管理员 WHERE(用户名='" + loginUserName + "' AND 密码='" + loginPassword + "')";
else
    loginQuery = "SELECT * FROM 学生基本信息表 WHERE(学号='" + loginUserName + "' AND 密码='" + loginPassword + "')";
```

3. 执行查询并验证结果:

```
loginResultSet = loginStatement.executeQuery(loginQuery);
boolean Records = loginResultSet.next();
if ( ! Records ) {
    JOptionPane.showMessageDialog(LoginFrame.this, "没有此用户或密码错误");
    return;
} else {
    login = 1; // 登录成功标记
}
```

### 4. 登录结果处理

1. 验证成功:

- `login` 变量被设为 1
- 根据用户类型打开对应的功能窗口:

java

```

if(selectedItem.equals("学生")) {
    JFrame f = new StudentsFrame();
    f.setVisible(true);
    dispose(); // 关闭当前登录窗口
} else if(selectedItem.equals("教师")) {
    JFrame f = new TeacherFrame();
    f.setVisible(true);
    dispose();
} else if(selectedItem.equals("管理员")) {
    JFrame f = new ManagerFrame();
    f.setVisible(true);
    dispose();
}

```

## 2. 验证失败:

- 显示 "没有此用户或密码错误" 的提示对话框
- 保持登录窗口打开, 允许用户重新输入

# 尝试SQL注入攻击

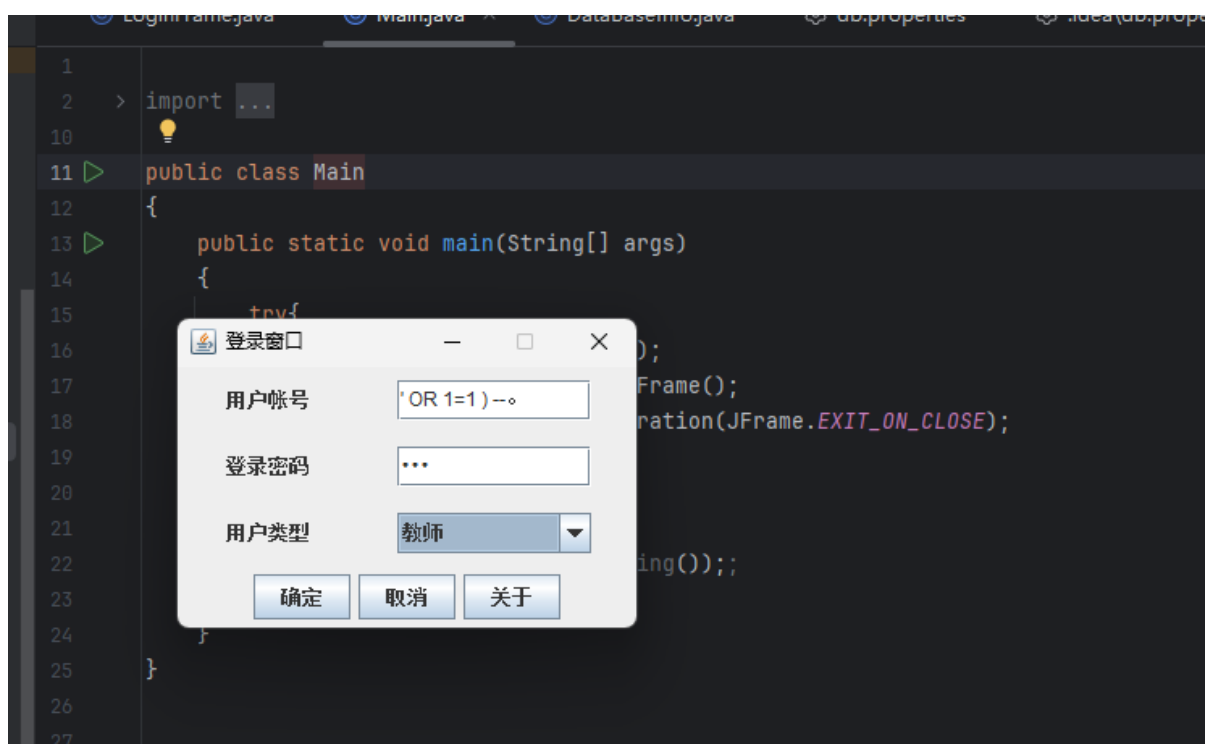
根据对 LoginFrame.java 登录逻辑的分析, 程序通过字符串拼接用户输入来构建SQL查询语句, 这导致了典型的SQL注入漏洞。本实验尝试以教师身份进行SQL注入攻击, 绕过正常的账号密码验证。

## 攻击步骤:

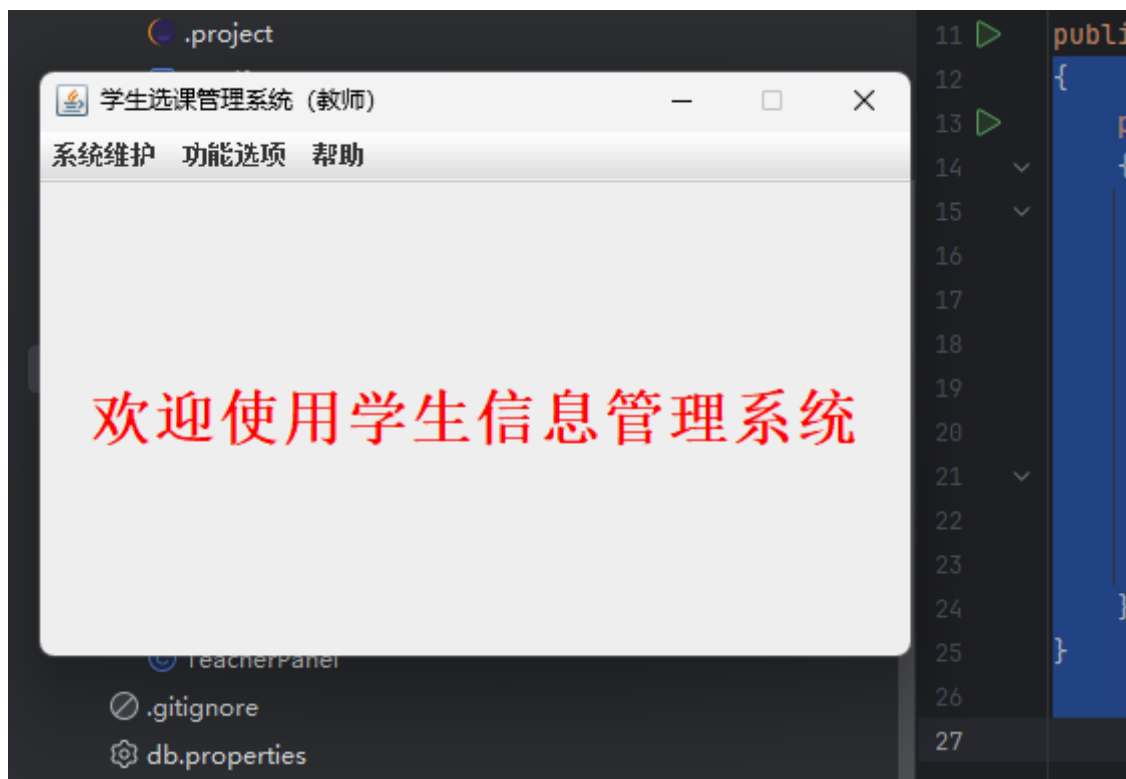
在用户账号输入注入载荷: ' OR 1=1 )--

登录密码输入任意内容

点击确定



登录成功



分析漏洞逻辑：当用户类型选择为“教师”，且“用户帐号”输入 ' OR 1=1 ) --，“登录密码”输入123时，LoginFrame.java 中构建的SQL查询语句（以教师登录为例，原始形式为 `SELECT * FROM 教师表 WHERE(登陆帐号=' ' + loginUserName + ' ' AND 登陆密码 = ' ' + loginPassword + ' ')`）在数据库中实际执行时会变成如下有效逻辑：

```
SELECT * FROM 教师表 WHERE(登陆帐号=' ' OR 1=1 )
```

这条SQL语句的解析过程如下：

1. 注入载荷中的第一个单引号 ' 闭合了原始语句中 登陆帐号=' 后面的单引号。
2. OR 1=1 引入了一个恒为真的条件。由于 1=1 永远为真，这个条件将使整个 WHERE 子句的逻辑结果趋向于真。
3. 紧随其后的右括号 ) 巧妙地闭合了原始 WHERE 子句最外层的左括号 (。
4. 最后的双破折号 -- 是SQL Server的单行注释符。它将从此处开始到行尾的所有内容（包括原始语句中剩余的 ' AND 登陆密码 ='123'）部分）都注释掉，使其不再 被数据库解析为有效的SQL代码。

因此，实际执行的 WHERE 子句简化为 (登陆帐号=" OR 1=1 )。由于 优先级低于 AND, 且 1=1 始终为真，无论 OR 逻辑运算符的 登陆帐号=" 的结果如何，整个 WHERE 子句 的评估结果都将是 TRUE。这使得数据库返回了 教师表 中的记录，ResultSet.next() 方法成功找到记录，从而绕过了正常的身份验证，实现了未授权登录。