

第四讲 电商数据库设计与 JDBC

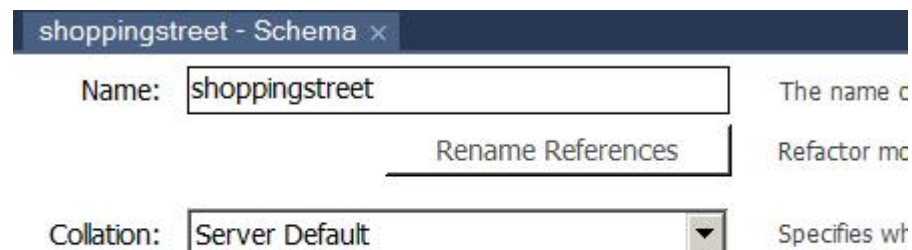
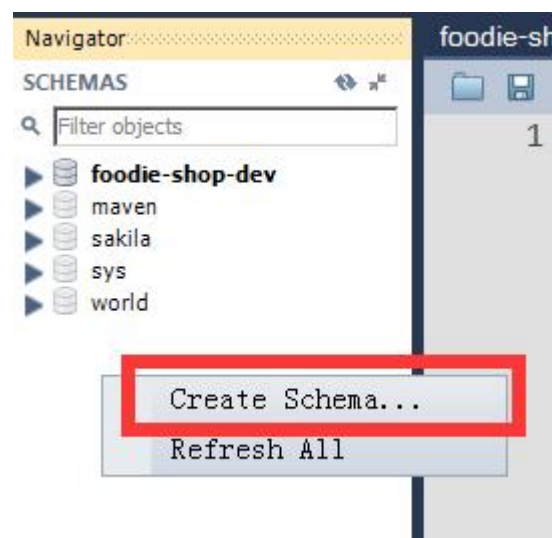
一、数据库设计

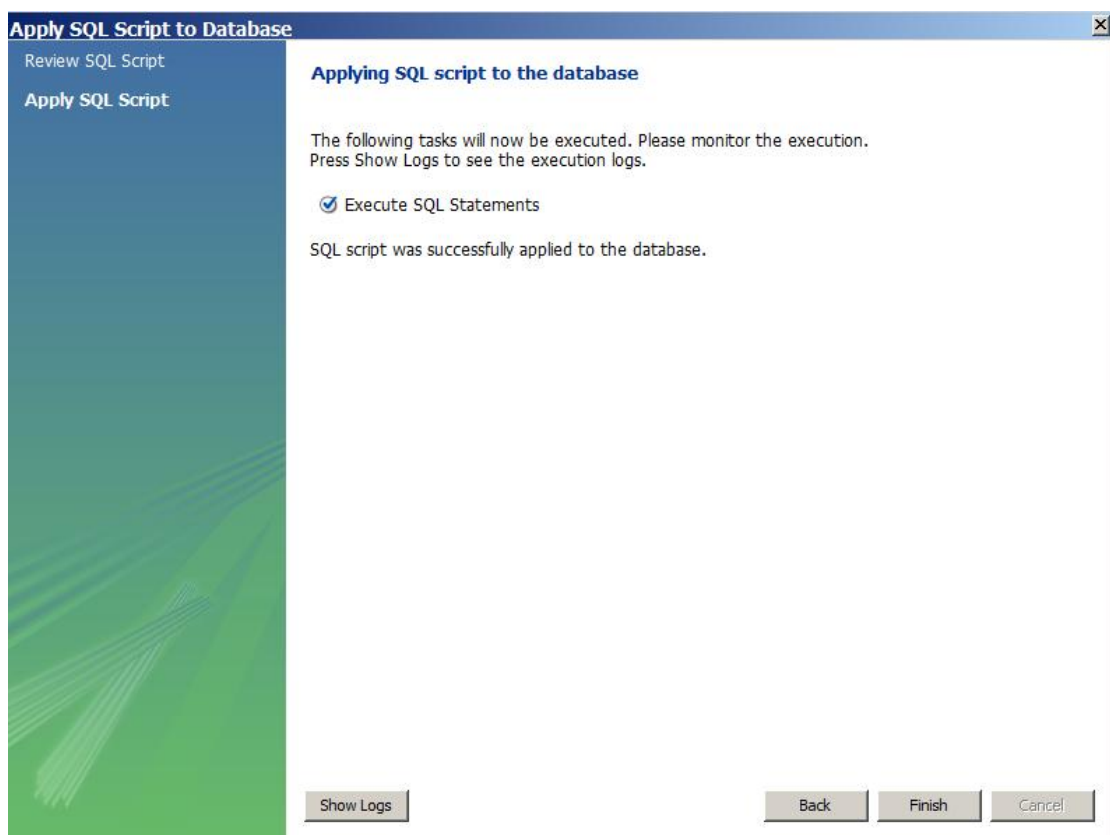
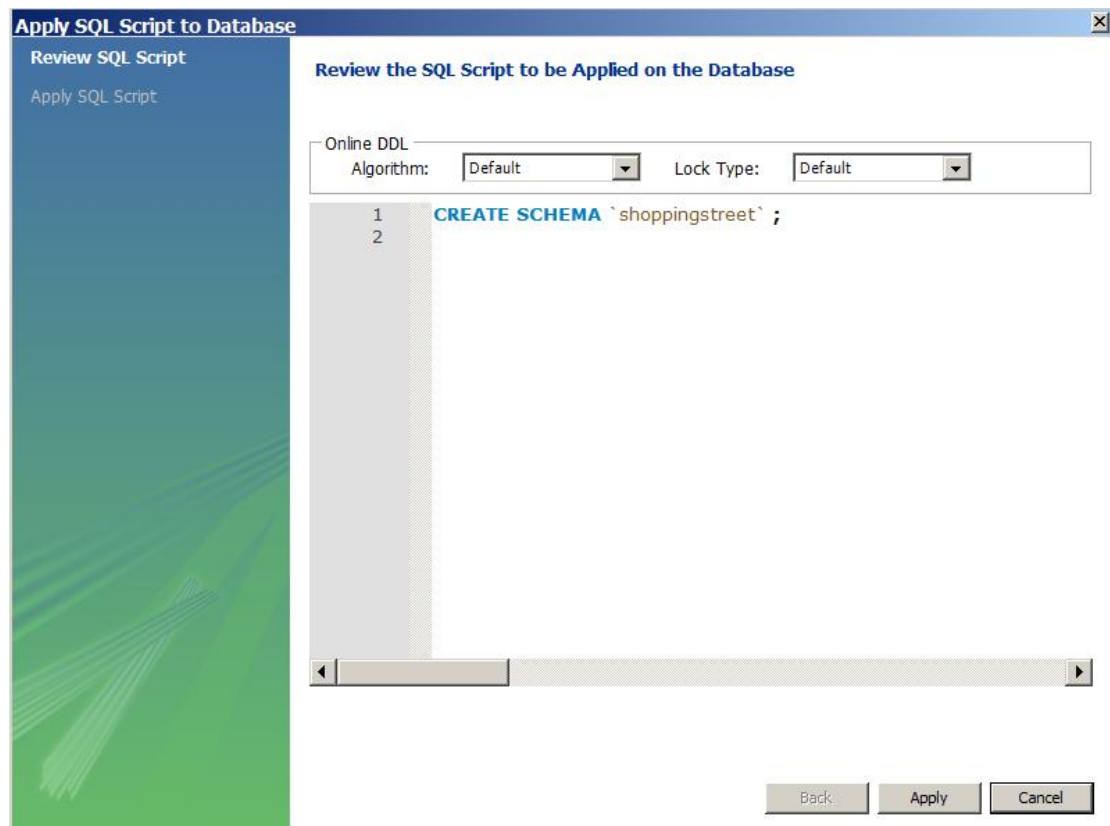
1. 用户购物业务线

用户 商品 订单

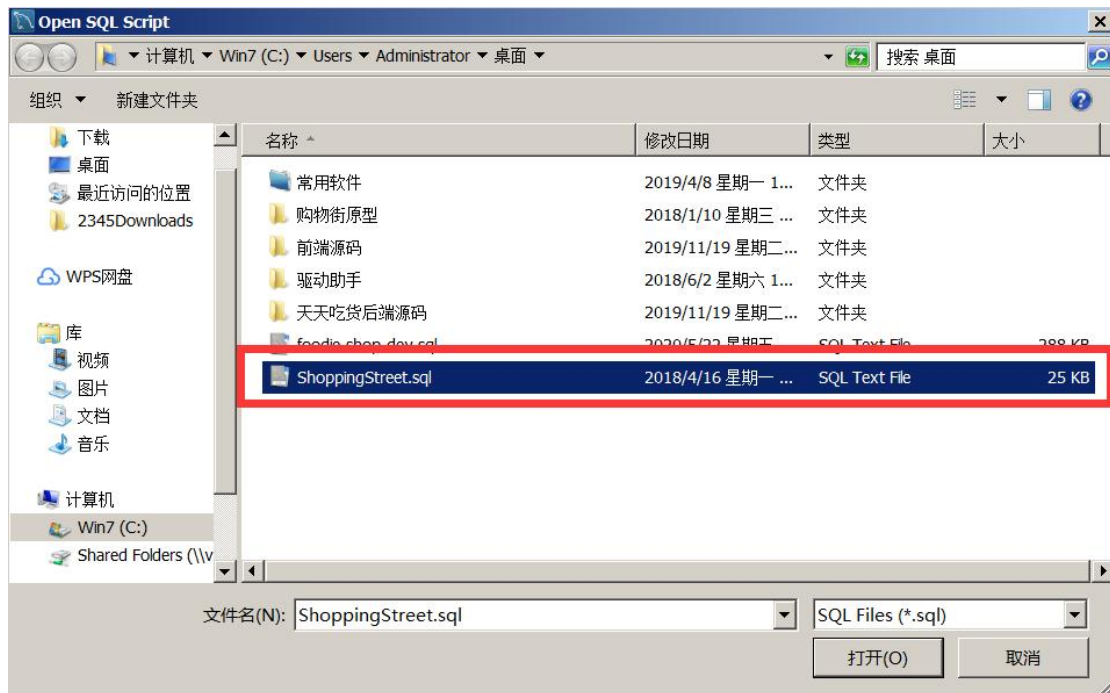
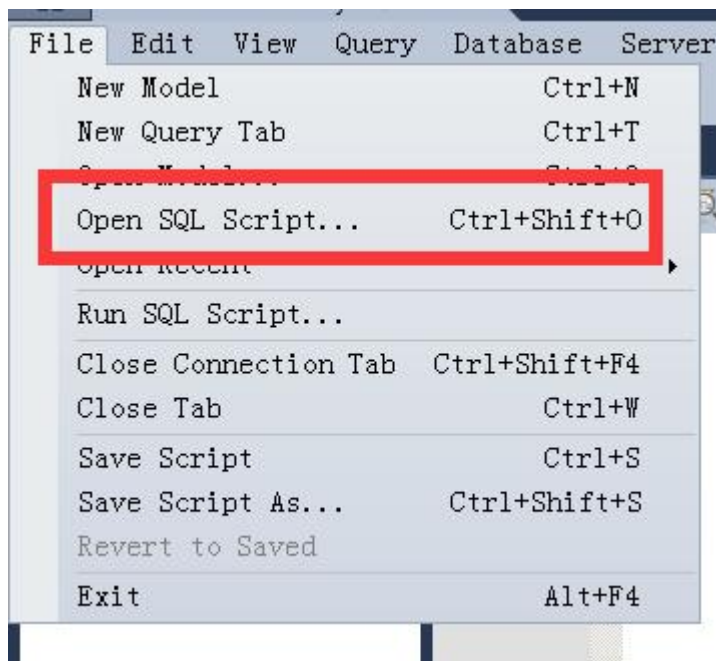
2. 导入 sql 脚本文件

在 workbench 中创建数据库 shoppingstreet

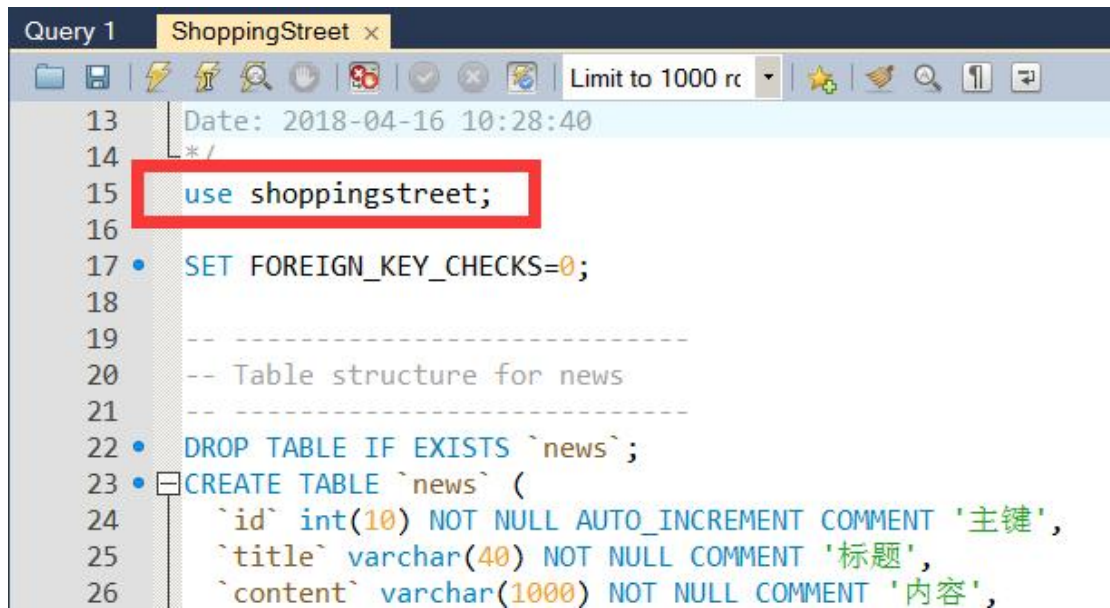




打开 sql 脚本文件



打开文件后，发现缺少对应数据库的引用

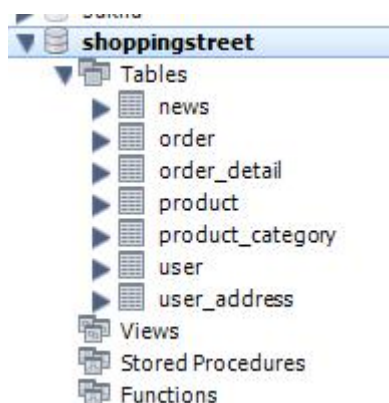


```
Query 1 ShoppingStreet x
Limit to 1000 rc
Date: 2018-04-16 10:28:40
*/
15 use shoppingstreet;
16
17 SET FOREIGN_KEY_CHECKS=0;
18
19 -----
20 -- Table structure for news
21 -----
22 DROP TABLE IF EXISTS `news`;
23 CREATE TABLE `news` (
24   `id` int(10) NOT NULL AUTO_INCREMENT COMMENT '主键',
25   `title` varchar(40) NOT NULL COMMENT '标题',
26   `content` varchar(1000) NOT NULL COMMENT '内容',
```

保存后，点击工具栏刷新按钮

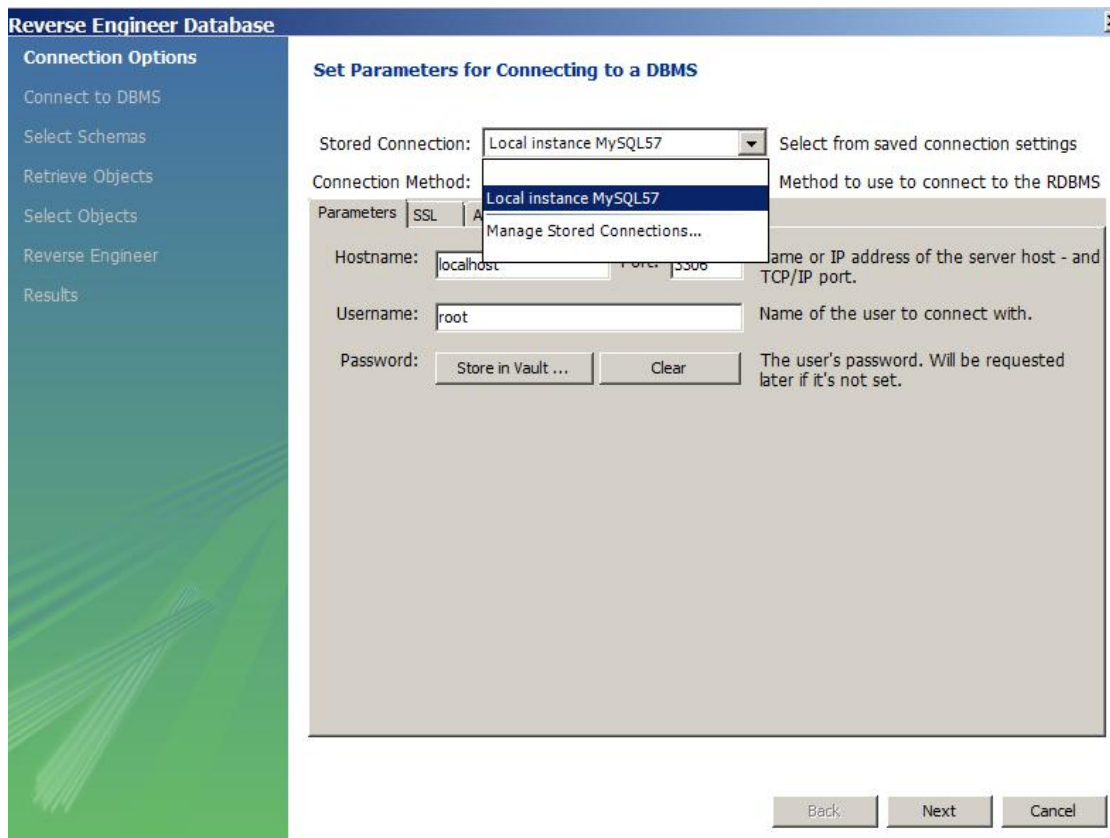
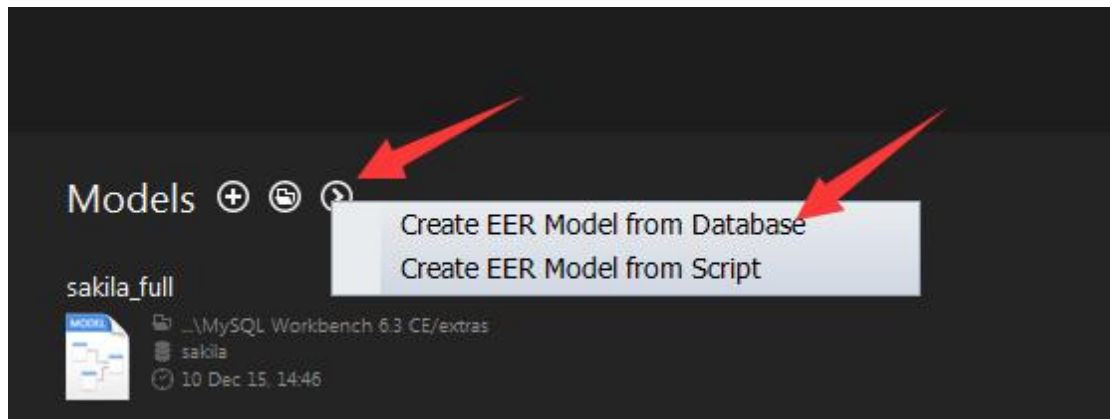


展开左侧数据库树，查看表是否被引入



3. workbench 逆向生成数据库模型

数据库中查看表结构不方便，可以在 workbench 中逆向生成数据库模型



Reverse Engineer Database

Connection Options

Connect to DBMS

Select Schemas


Retrieve Objects

Select Objects

Reverse Engineer

Results

Select Schemas to Reverse Engineer

 Select the schemas below you want to include:

☐ foodie-shop-dev

☐ maven

☐ sakila

☒ shoppingstreet

☐ sys

☐ world

Back

Next

Cancel

Reverse Engineer Database

Connection Options

Connect to DBMS

Select Schemas


Retrieve Objects

Select Objects

Reverse Engineer

Results

Select Objects to Reverse Engineer



☒ Import MySQL Table Objects

Show Filter

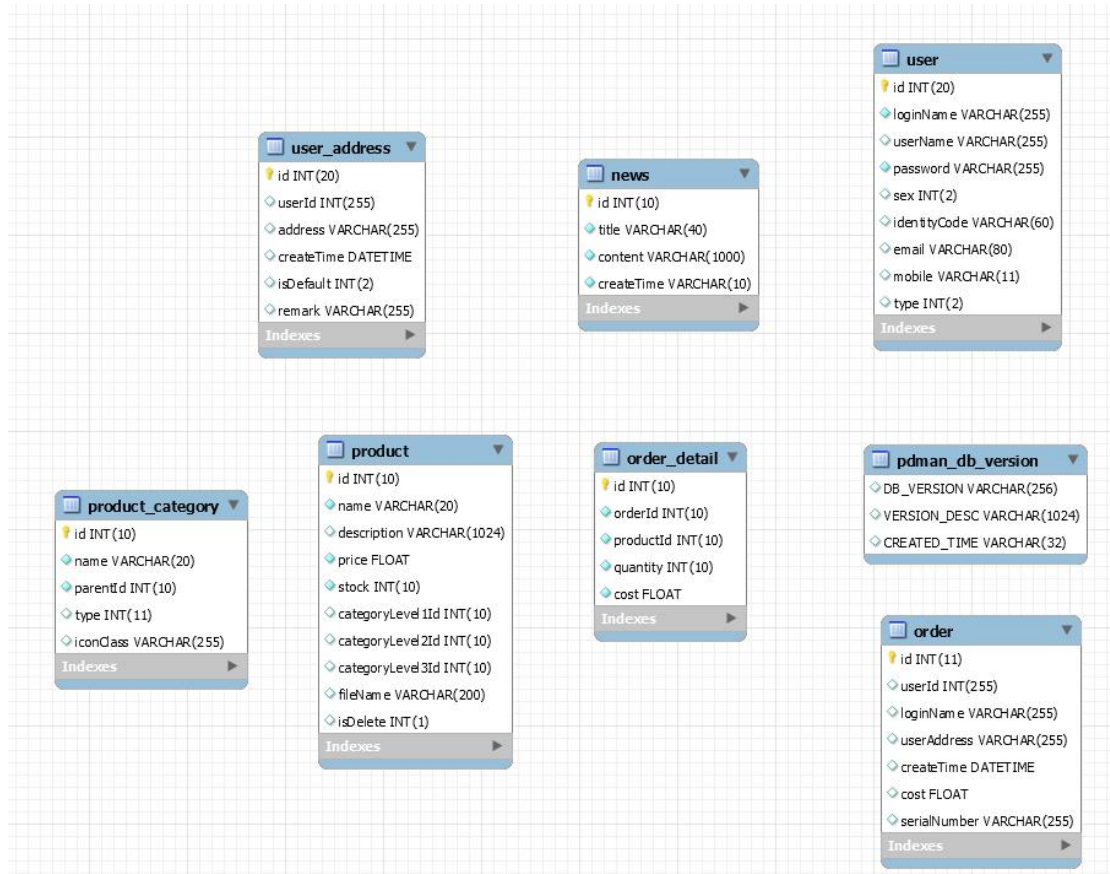
8 Total Objects, 8 Selected

☒ Place imported objects on a diagram

Back

Execute >

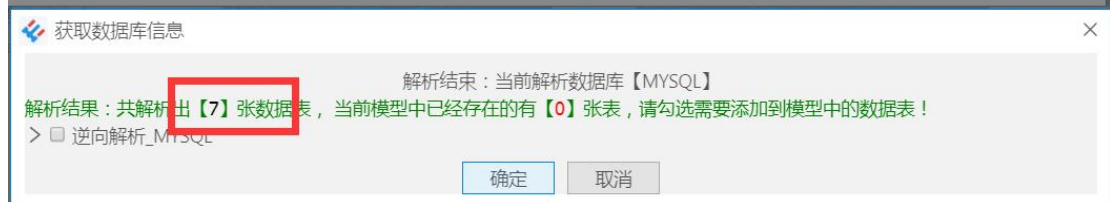
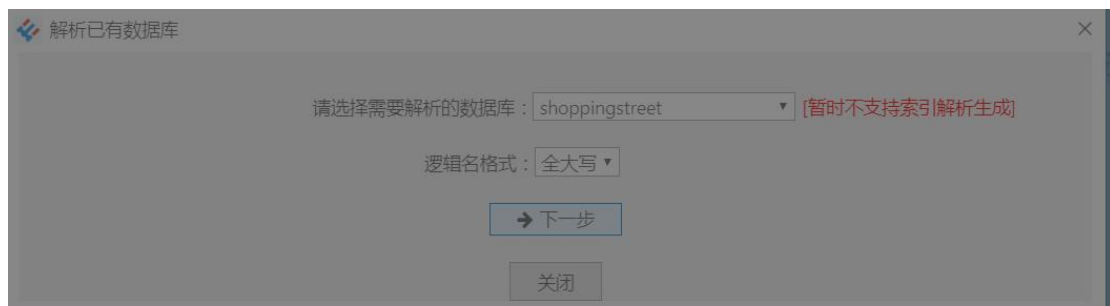
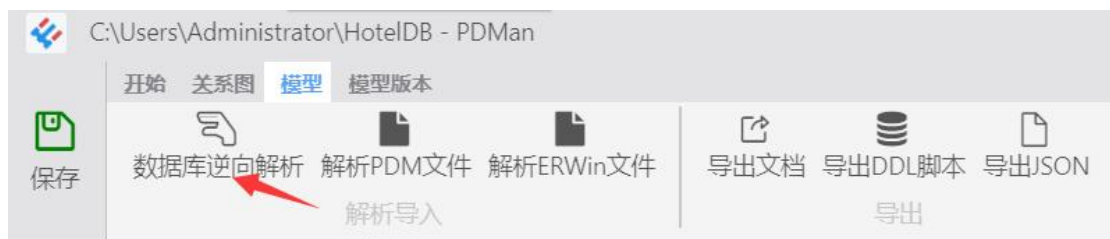
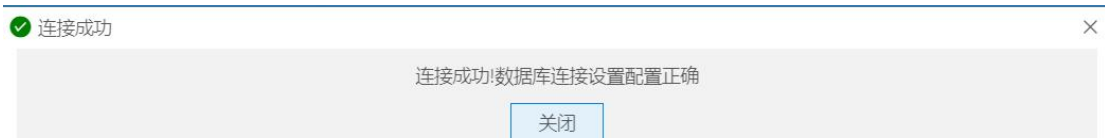
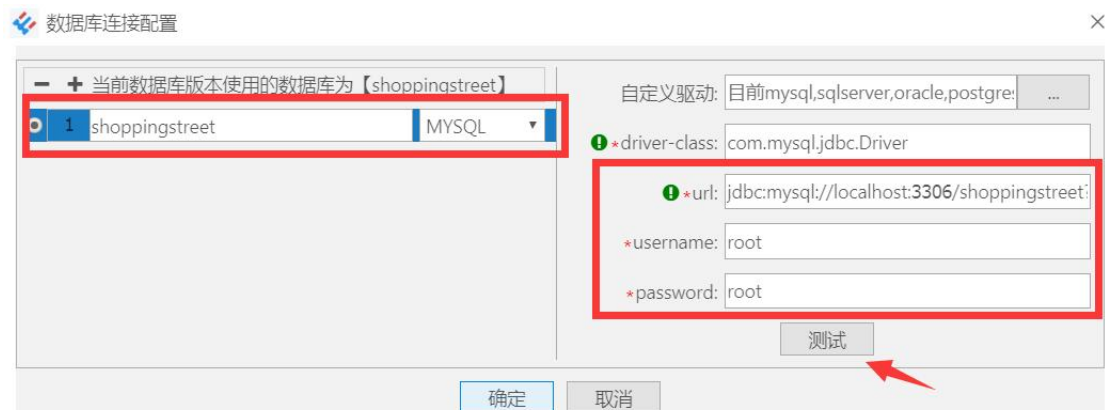
Cancel



4. PDMan 中逆向生成数据库

不方便查看 字段注释，可以在 PDMan 中逆向生成数据库





将表拖入到关系图视图中

ORDER_DETAIL[]				
主键	ID	INT_10	<PK>	
订单主键	ORDERID	INT_10		
商品主键	PRODUCTID	INT_10		
数量	QUANTITY	INT_10		
消费	COST	FLOAT_12		

ORDER_DETAIL[] 1				
主键	ID	INT_10	<PK>	
订单主键	ORDERID	INT_10		
商品主键	PRODUCTID	INT_10		
数量	QUANTITY	INT_10		
消费	COST	FLOAT_12		

USER[]				
主键	ID	INT_10	<PK>	
登录名	LOGINNAME	VARCHAR_255		
用户名	USERNAME	VARCHAR_255		
密码	PASSWORD	VARCHAR_255		
性别(1:男 0:女)	SEX	INT_10		
身份证号	IDENTITYCODE	VARCHAR_60		
邮箱	EMAIL	VARCHAR_80		
手机	MOBILE	VARCHAR_11		
类型 (1: 后台 0:前台)	TYPE	INT_10		

PRODUCT[]				
主键	ID	INT_10	<PK>	
名称	NAME	VARCHAR_20		
描述	DESCRIPTION	VARCHAR_1024		
价格	PRICE	FLOAT_12		
库存	STOCK	INT_10		
分类1	CATEGORYLEVEL1ID	INT_10		
分类2	CATEGORYLEVEL2ID	INT_10		
分类3	CATEGORYLEVEL3ID	INT_10		
文件名称	FILENAME	VARCHAR_200		
是否删除(1: 删除 0: 未删除)	ISDELETE	INT_10		

USER_ADDRESS[]				
主键id	ID	INT_10	<PK>	
用户主键	USERID	INT_10		
地址	ADDRESS	VARCHAR_255		
创建时间	CREATETIME	DATETIME		
是否是默认地址 (1:是 0:否)	ISDEFAULT	INT_10		
备注	REMARK	VARCHAR_255		

PRODUCT_CATEGORY[]				
主键	ID	INT_10	<PK>	
名称	NAME	VARCHAR_20		
父级目录id	PARENTID	INT_10		
级别(1:一级 2: 二级 3: 三级)	TYPE	INT_10		
图标	ICONCLASS	VARCHAR_255		

5. 业务相关 sql:

// 保存订单

```
insert                                     into
shoppingStreet.order(userId,loginName,userAddress,createTime,cost,serialNumber)
values(?,?,?,?,?,?)
```

// 保存订单详情

```
insert into order_detail(orderId,productId,quantity,cost) values(?,?,?,?)
```

//根据名称搜索商品

select

```
id,name,description,price,stock,categoryLevel1Id,categoryLevel2Id,categoryLevel3I
d,fileName,isDelete from product  where name like ?
```

//根据 id 搜索商品

新增用户

```
insert into user(loginName,password,email,mobile) values(?,?,?,?)
```

登录

```
select id,loginName,password,email,mobile from user where loginName=? and
password=?
```

二、jdbc

1、什么是 JDBC

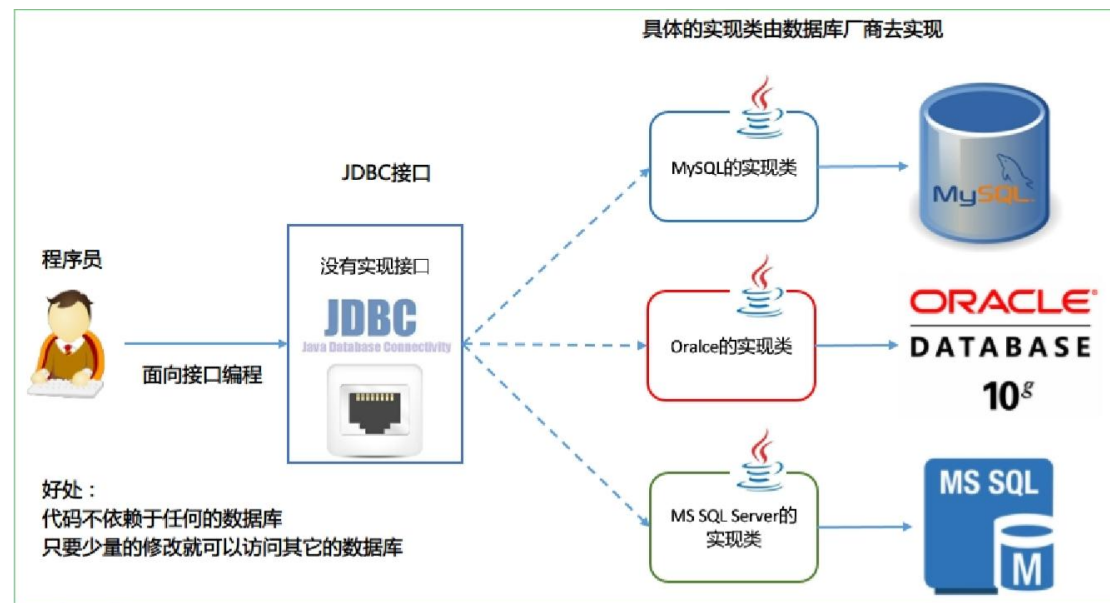
JDBC 规范定义接口，具体的实现由各大数据库厂商来实现。

JDBC 是 Java 访问数据库的标准规范，真正怎么操作数据库还需要具体的实现类，也就

是数据库驱动。每个数据库厂商根据自家数据库的通信格式编写好自己数据库的驱动。所以我们只需要会调用 JDBC 接口中的方法即可，数据库驱动由数据库厂商提供。

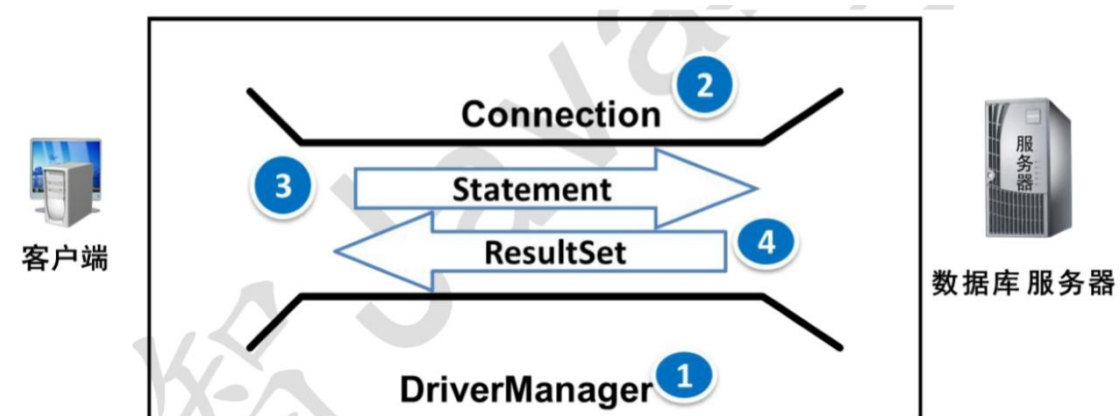
使用 JDBC 的好处：

- 程序员如果要开发访问数据库的程序，只需要会调用 JDBC 接口中的方法即可，不用关注类是如何实现的。
- 使用同一套 Java 代码，进行少量的修改就可以访问其他 JDBC 支持的数据库



2、JDBC 访问数据库的步骤

① 步骤



1) 注册和加载驱动(可以省略)

- 2) 获取连接
- 3) Connection 获取 Statement 对象
- 4) 使用 Statement 对象执行 SQL 语句
- 5) 返回结果集
- 6) 释放资源

② JDBC 核心 API

java.sql 包含 所有与 JDBC 访问数据库相关的接口和类

接口或类	作用
DriverManager 类	1) 管理和注册数据库驱动 2) 得到数据库连接对象
Connection 接口	一个连接对象，可用于创建 Statement 和 PreparedStatement 对象
Statement 接口	一个 SQL 语句对象，用于将 SQL 语句发送给数据库服务器
PreparedStatement 接口	一个 SQL 语句对象，是 Statement 的子接口
ResultSet 接口	用于封装数据库查询的结果集，返回给客户端 Java 程序

3、执行 DDL 操作

- 需求：使用 JDBC 在 MySQL 的数据库中创建一张学生表

Field	Type		Null	Key	Default		Extra
id	int(11)	7B	NO	PRI	(NULL)	OK	auto_increment
name	varchar(20)	11B	NO		(NULL)	OK	
gender	tinyint(1)	10B	YES		1	1B	
birthday	date	4B	YES		(NULL)	OK	

- 代码

```
package club.semicircle;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

/** * 创建一张学生表 */
public class Demo4DDL {
    public static void main(String[] args) {
```

```
//1. 创建连接
Connection conn = null;
Statement statement = null;
try {
    conn = DriverManager.getConnection("jdbc:mysql:///day24", "root",
"root");

    //2. 通过连接对象得到语句对象
    statement = conn.createStatement();
    //3. 通过语句对象发送 SQL 语句给服务器
    //4. 执行 SQL
    statement.executeUpdate("create table student (id int PRIMARY key
auto_increment, " +
        "name varchar(20) not null, gender boolean, birthday date)");
    //5. 返回影响行数(DDL 没有返回值)
    System.out.println("创建表成功");
} catch (SQLException e) {
    e.printStackTrace();
}
//6. 释放资源
finally {
    //关闭之前要先判断
    if (statement != null) {
        try {
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
}  
}
```

4、加载和注册驱动

Class.forName(数据库驱动实现类) 加载和注册数据库驱动，数据库驱动由 mysql 厂商
"com.mysql.jdbc.Driver"

5、DriverManager 类

① DriverManager 作用

- 管理和注册驱动
- 创建数据库的连接

② 类中的方法

DriverManager 类中的静态方法	描述
Connection getConnection (String url, String user, String password)	通过连接字符串，用户名，密码来得到数据库的连接对象
Connection getConnection (String url, Properties info)	通过连接字符串，属性对象来得到连接对象

③ 使用 JDBC 连接数据库的四个参数

JDBC 连接数据库的四个参数	说明
用户名	登录的用户名
密码	登录的密码
连接字符串 URL	不同的数据库 URL 是不同的，mysql 的写法 jdbc:mysql://localhost:3306/数据库[?参数名=参数值]

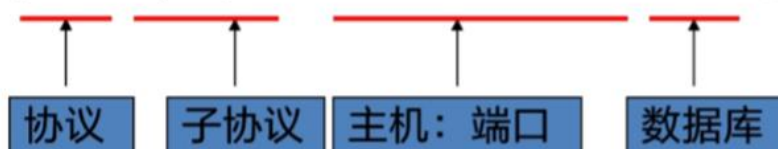
驱动类的字符串名	com.mysql.jdbc.Driver
----------	-----------------------

- 连接数据库的 URL 地址格式：

协议名:子协议://服务器名或 IP 地址:端口号/数据库名?参数=参数值

- URL用于标识数据库的位置，程序员通过URL地址告诉JDBC程序连接哪个数据库，URL的写法为：

jdbc:mysql: //localhost:3306/test ? 参数名=参数值



乱码的处理 如果数据库出现乱码，可以指定参数: ?characterEncoding=utf8，表示让数据库以 UTF-8 编码来处理数据。

jdbc:mysql://localhost:3306/数据库?characterEncoding=utf8

举例：获取 MySQL 的数据库连接对象

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

/** * 得到连接对象
 */
public class Demo2 {
    public static void main(String[] args) throws SQLException {
        String url = "jdbc:mysql://localhost:3306/day24";
        //1) 使用用户名、密码、 URL 得到连接对象
        Connection connection = DriverManager.getConnection(url, "root", "root");
        //com.mysql.jdbc.JDBC4Connection@68de145
        System.out.println(connection);
    }
}
```

6、Connection 接口

① Connection 作用：

Connection 接口，具体的实现类由数据库的厂商实现，代表一个连接对象。

② Connection 方法：

Statement createStatement() 创建一条 SQL 语句对象

7、Statement 接口

Statement 作用

代表一条语句对象，用于发送 SQL 语句给服务器，用于执行静态 SQL 语句并返回它所生成结果的对象。

Statement 中的方法

Statement 接口中的方法	描述
int executeUpdate(String sql)	用于发送 DML 语句，增删改的操作，insert、update、delete 参数：SQL 语句 返回值：返回对数据库影响的行数
ResultSet executeQuery(String sql)	用于发送 DQL 语句，执行查询的操作。select 参数：SQL 语句 返回值：查询的结果集

8、释放资源

- 1) 需要释放的对象：ResultSet 结果集，Statement 语句，Connection 连接
- 2) 释放原则：先开的后关，后开的先关。ResultSet -> Statement -> Connection

3) 放在哪个代码块中: finally 块

9、执行 DML 操作

- 需求: 向学生表中添加 4 条记录, 主键是自动增长

id	name	gender	birthday
1	孙悟空	1	1993-03-24
2	白骨精	0	1995-03-24
3	猪八戒	1	1903-03-24
4	嫦娥	0	1993-03-11

- 代码:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

/** * 向学生表中添加 4 条记录, 主键是自动增长 */
public class Demo5DML {
    public static void main(String[] args) throws SQLException {
        //      1) 创建连接对象
        Connection connection =
DriverManager.getConnection("jdbc:mysql:///day24", "root", "root");
        //      2) 创建 Statement 语句对象
        Statement statement = connection.createStatement();
        //      3) 执行 SQL 语句: executeUpdate(sql)
        int count = 0;
        //      4) 返回影响的行数
        count += statement.executeUpdate("insert into student values(null, '孙悟空', 1, '1993-0324')");
        count += statement.executeUpdate("insert into student values(null, '白骨精', 0, '1995-0324')");
```

```

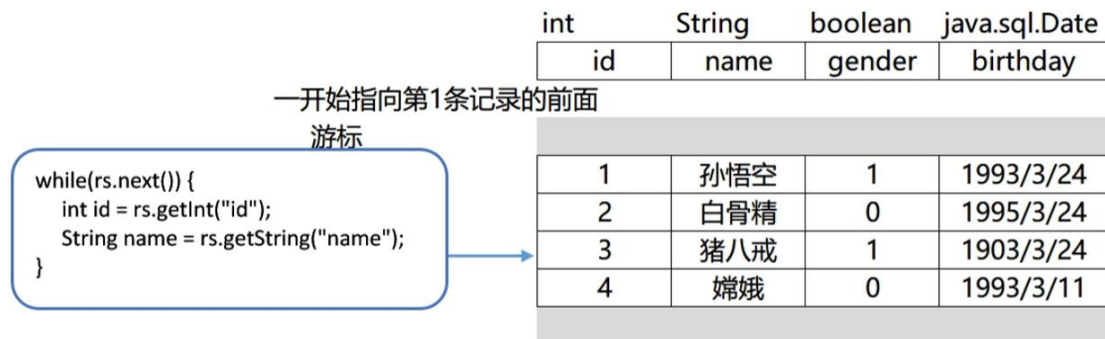
        count += statement.executeUpdate("insert into student values(null, '猪八戒', 1, '1903-0324')");
        count += statement.executeUpdate("insert into student values(null, '嫦娥', 0, '1993-0311')");
        System.out.println("插入了" + count + "条记录");
//        5) 释放资源
        statement.close();
        connection.close();
    }
}

```

10、执行 DQL 操作

① ResultSet 接口：

作用：封装数据库查询的结果集，对结果集进行遍历，取出每一条记录。



② 接口中的方法：

ResultSet 接口中的方法	描述
boolean next()	1) 游标向下移动 1 行 2) 返回 boolean 类型，如果还有下一条记录，返回 true，否则返回 false 数据类型
getXxx()	1) 通过字段名，参数是 String 类型。返回不同的类型 2) 通过列号，参数是整数，从 1 开始。返回不同的类型

③ 举例：查询所有的学员信息

```
import java.sql.*;

/** * 查询所有的学生信息 */
public class Demo6DQL {
    public static void main(String[] args) throws SQLException {
        //1) 得到连接对象
        Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/day24","root","root");
        //2) 得到语句对象
        Statement statement = connection.createStatement();
        //3) 执行 SQL 语句得到结果集 ResultSet 对象
        ResultSet rs = statement.executeQuery("select * from student");
        //4) 循环遍历取出每一条记录
        while(rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            //5) 输出的控制台上
            System.out.println("编号:" + id + ", 姓名:" + name + ", 性别:" + gender
+ ", 生日:" + birthday);
        }
        //6) 释放资源
        rs.close();
        statement.close();
        connection.close();
    }
}
```

④ 关于 ResultSet 接口中的注意事项：

- 1) 如果光标在第一行之前，使用 rs.getXX()获取列值，报错：Before start of result set
- 2) 如果光标在最后一行之后，使用 rs.getXX()获取列值，报错：After end of result set
- 3) 使用完毕以后要关闭结果集 ResultSet，再关闭 Statement，再关闭 Connection

11、数据库工具类 JdbcUtils

创建类 JdbcUtil 包含 3 个方法：

- 1) 可以把几个字符串定义成常量：用户名，密码，URL，驱动类
- 2) 得到数据库的连接：getConnection()
- 3) 关闭所有打开的资源：close(Connection conn, Statement stmt) , close(Connection conn, Statement stmt, ResultSet rs)

```
import java.sql.*;

/**
 * 访问数据库的工具类
 */
public class JdbcUtils {

    //可以把几个字符串定义成常量：用户名，密码，URL，驱动类
    private static final String USER = "root";
    private static final String PWD = "root";
    private static final String URL = "jdbc:mysql://localhost:3306/day24";
    private static final String DRIVER= "com.mysql.jdbc.Driver";

    /**      * 注册驱动      */
    static {
        try {
            Class.forName(DRIVER);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    /**      * 得到数据库的连接      */
    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL,USER,PWD);
    }

    /**      * 关闭所有打开的资源      */
    public static void close(Connection conn, Statement stmt) {
        if (stmt!=null) {
```



```

        try {
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (conn!=null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
/**      * 关闭所有打开的资源      */
public static void close(Connection conn, Statement stmt, ResultSet rs) {
    if (rs!=null) {
        try {
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    close(conn, stmt);
}
}

```

举例：用户登录

sql 脚本

```

create table user (
    id int primary key auto_increment,
    name varchar(20),
    password varchar(20)
)
insert into user values (null,'jack','123'),(null,'rose','456');

```

```
-- 登录
select * from user where name='JACK' and password='123';

-- 登录失败
select * from user where name='JACK' and password='333';
```

java 代码:

```
import com.itheima.utils.JdbcUtils;
import javax.xml.transform.Result;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class Demo7Login {

    //从控制台上输入的用户名和密码
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入用户名: ");
        String name = sc.nextLine();
        System.out.println("请输入密码: ");
        String password = sc.nextLine();
        login(name, password);
    }

    /**      * 登录的方法      */
    public static void login(String name, String password) {
        //a) 通过工具类得到连接
        Connection connection = null;
        Statement statement = null;
        ResultSet rs = null;
        try {
            connection = JdbcUtils.getConnection();
```

```

        //b) 创建语句对象，使用拼接字符串的方式生成 SQL 语句
        statement = connection.createStatement();
        //c) 查询数据库，如果有记录则表示登录成功，否则登录失败
        String sql = "select * from user where name='" + name + "' and
password='" + password + "'";
        System.out.println(sql);
        rs = statement.executeQuery(sql);
        if (rs.next()) {
            System.out.println("登录成功，欢迎您：" + name);
        } else {
            System.out.println("登录失败");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        //d) 释放资源
        JdbcUtils.close(connection, statement, rs);
    }
}
}
}

```

- SQL 注入问题

当我们输入以下密码，我们发现我们账号和密码都不对竟然登录成功了

请输入用户名：

newboy

请输入密码：

a' or '1'='1

select * from user where name='newboy' and password='a' or '1'='1'

登录成功，欢迎您：newboy

问题分析：

select * from user where name='newboy' and password='a' or '1'='1'

name='newboy' and password='a' 为假

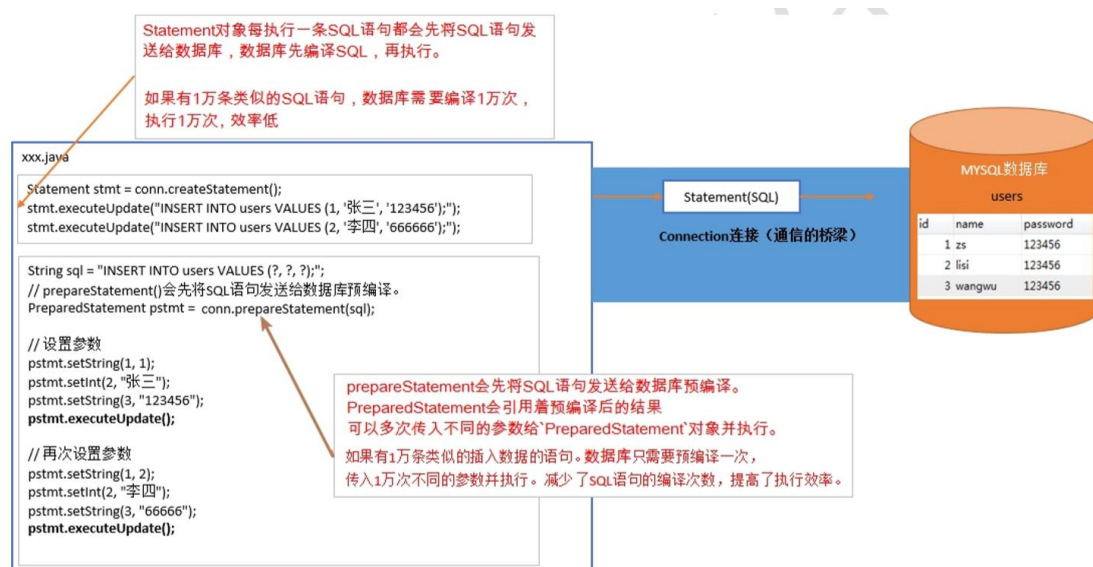
'1'='1' 真 相当于

select * from user where true; 查询了所有记录

12、PreparedStatement 接口

PreparedStatement 是 Statement 接口的子接口，继承于父接口中所有的方法。它是一个预编译的 SQL 语句

PreparedStatement 的执行原理



PreparedStatement 的好处

- prepareStatement() 会先将 SQL 语句发送给数据库预编译。PreparedStatement 会引用着预编译后的结果。可以多次传入不同的参数给 PreparedStatement 对象并执行。减少 SQL 编译次数，提高效率。
- 安全性更高，没有 SQL 注入的隐患。
- 提高了程序的可读性

使用 PreparedStatement 的步骤

- 1) 编写 SQL 语句，未知内容使用?占位："SELECT * FROM user WHERE name=? AND

password=?";

2) Connection 创建 PreparedStatement 对象

PreparedStatement prepareStatement(String sql) 指定预编译的 SQL 语句, SQL 语句中使用占位符? 创建一个语句对象

3) 设置实际参数: setXxx(占位符的位置, 真实的值)

4) 执行参数化 SQL 语句

5) 关闭资源

```
import com.itheima.utils.JdbcUtils;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

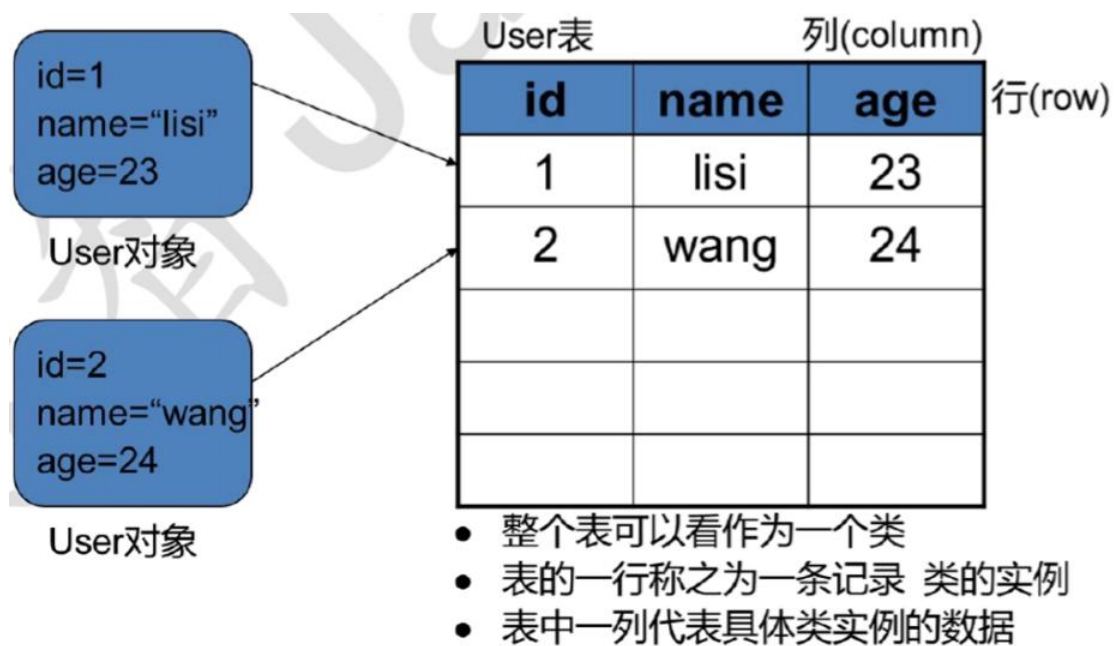
/** * 使用 PreparedStatement */
public class Demo8Login {
    //从控制台上输入的用户名和密码
    public static void main(String[] args) throws SQLException {
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入用户名: ");
        String name = sc.nextLine();
        System.out.println("请输入密码: ");
        String password = sc.nextLine();
        login(name, password);
    }
    /**
     * 登录的方法
     * @param name
     * @param password
     */
    private static void login(String name, String password) throws SQLException {
        Connection connection = JdbcUtils.getConnection();
        //写成登录 SQL 语句, 没有单引号
        String sql = "select * from user where name=? and password=?";
        //得到语句对象
        PreparedStatement ps = connection.prepareStatement(sql);
        //设置参数
```

```

        ps.setString(1, name);
        ps.setString(2,password);
        ResultSet resultSet = ps.executeQuery();
        if (resultSet.next()) {
            System.out.println("登录成功: " + name);
        }
        else {
            System.out.println("登录失败");
        }
        //释放资源,子接口直接给父接口
        JdbcUtils.close(connection,ps,resultSet);
    }
}

```

13、表与类的关系



- 案例 1：使用 PreparedStatement 查询一条数据，封装成一个学生 Student 对象

```

import com.itheima.entity.Student;
import com.itheima.utils.JdbcUtils;
import java.sql.Connection;
import java.sql.PreparedStatement;

```



```

import java.sql.ResultSet;
import java.sql.SQLException;

public class Demo9Student {
    public static void main(String[] args) throws SQLException {
        //创建学生对象
        Student student = new Student();
        Connection connection = JdbcUtils.getConnection();
        PreparedStatement ps = connection.prepareStatement("select * from
student where id=?");
        //设置参数
        ps.setInt(1,2);
        ResultSet resultSet = ps.executeQuery();
        if (resultSet.next()) {
            //封装成一个学生对象
            student.setId(resultSet.getInt("id"));
            student.setName(resultSet.getString("name"));
            student.setGender(resultSet.getBoolean("gender"));
            student.setBirthday(resultSet.getDate("birthday"));
        }
        //释放资源
        JdbcUtils.close(connection,ps,resultSet);

        //可以数据
        System.out.println(student);
    }
}

```

- 案例 2: 将多条记录封装成集合 List<Student>, 集合中每个元素是一个 JavaBean 实体类

```

import com.itheima.entity.Student;
import com.itheima.utils.JdbcUtils;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

```

```

import java.util.ArrayList;
import java.util.List;
public class Demo10List {
    public static void main(String[] args) throws SQLException {
        //创建一个集合
        List<Student> students = new ArrayList<>();
        Connection connection = JdbcUtils.getConnection();
        PreparedStatement ps = connection.prepareStatement("select * from
student");
        //没有参数替换
        ResultSet resultSet = ps.executeQuery();
        while(resultSet.next()) {
            //每次循环是一个学生对象
            Student student = new Student();
            //封装成一个学生对象
            student.setId(resultSet.getInt("id"));
            student.setName(resultSet.getString("name"));
            student.setGender(resultSet.getBoolean("gender"));
            student.setBirthday(resultSet.getDate("birthday"));
            //把数据放到集合中
            students.add(student);
        }
        //关闭连接
        JdbcUtils.close(connection,ps,resultSet);
        //使用数据
        for (Student stu: students) {
            System.out.println(stu);
        }
    }
}

```

- PreparedStatement 执行 DML 操作

```

import com.itheima.utils.JdbcUtils;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

```

```

public class Demo11DML {
    public static void main(String[] args) throws SQLException {
        //insert();
        //update();
        delete();
    }

    //插入记录
    private static void insert() throws SQLException {
        Connection connection = JdbcUtils.getConnection();
        PreparedStatement ps = connection.prepareStatement("insert into
student values(null,?,?,?)");
        ps.setString(1,"小白龙");
        ps.setBoolean(2,true);
        ps.setDate(3,java.sql.Date.valueOf("1999-11-11"));
        int row = ps.executeUpdate();
        System.out.println("插入了" + row + "条记录");
        JdbcUtils.close(connection,ps);
    }

    //更新记录: 换名字和生日
    private static void update() throws SQLException {
        Connection connection = JdbcUtils.getConnection();
        PreparedStatement ps = connection.prepareStatement("update student
set name=?, birthday=? where id=?");
        ps.setString(1,"黑熊怪");
        ps.setDate(2,java.sql.Date.valueOf("1999-03-23"));
        ps.setInt(3,5);
        int row = ps.executeUpdate();
        System.out.println("更新" + row + "条记录");
        JdbcUtils.close(connection,ps);
    }

    //删除记录: 删除第 5 条记录
    private static void delete() throws SQLException {
        Connection connection = JdbcUtils.getConnection();

```

```
        PreparedStatement ps = connection.prepareStatement("delete from  
student where id=?");  
        ps.setInt(1,5);  
        int row = ps.executeUpdate();  
        System.out.println("删除了" + row + "条记录");  
        JdbcUtils.close(connection,ps);  
    }  
}
```