

# MySQL 多表&JDBC

## 今日内容介绍

- ◆ MySQL 多表
- ◆ 使用 JDBC 完成分类表 CRUD 的操作

## 今日内容学习目标

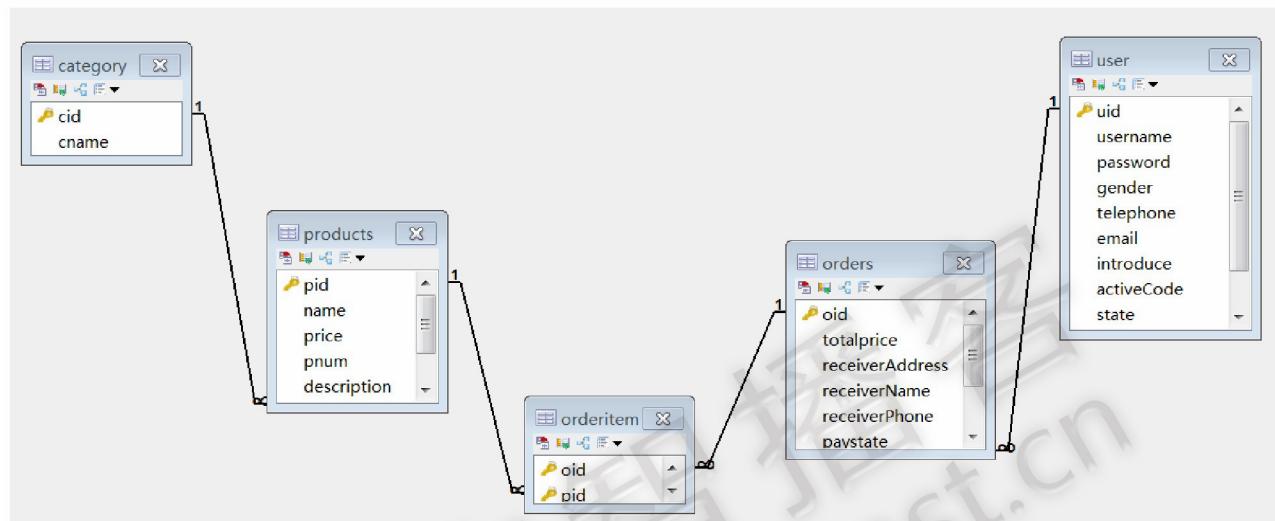
- ◆ 能够描述表与表的关系
- ◆ 能够独立编写一对多表关系 SQL 语句
- ◆ 能够独立编写多对多表关系 SQL 语句
- ◆ 能够使用 SQL 进行多表查询
- ◆ 能够使用 JDBC 发送 insert 语句完成单表【添加】操作
- ◆ 能够使用 JDBC 发送 update 语句完成单表【更新】操作
- ◆ 能够使用 JDBC 发送 delete 语句完成单表【删除】操作
- ◆ 能够使用 JDBC 发送 select 语句完成单表【查询】操作

# 第1章 商城系统表结构实现

## 1.1 需求分析

昨天我们已经回顾了单表操作，今天我们将真正的完成商城系统表结构的实现。

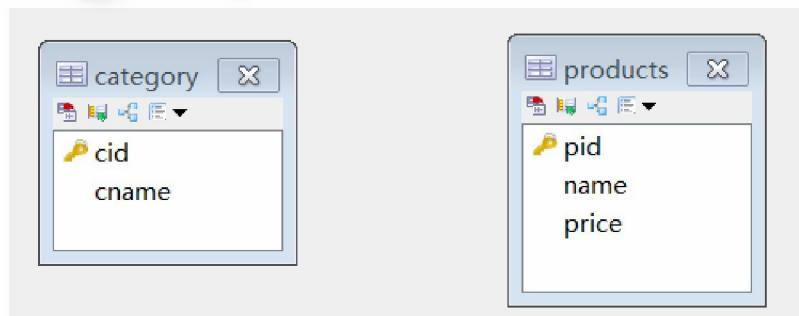
为了方便大家实现，现提供表关系图，如下：

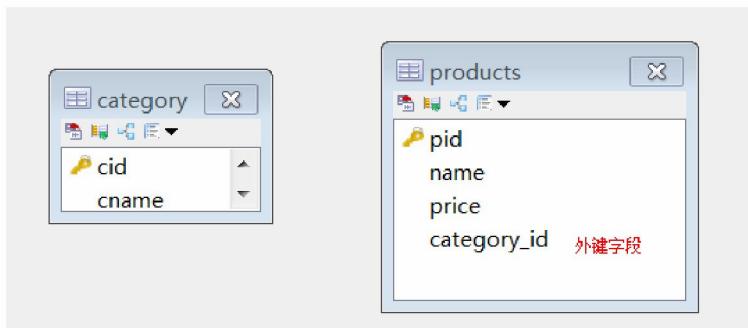


通过对上图内容的查看，整个案例中将使用 5 张表，且表与表之间存在一定的关系，接下来我们将一起学习 sql 的多表操作。

## 1.2 外键

现在我们有两张表“分类表”和“商品表”，为了表明商品属于哪个分类，通常情况下，我们将在商品表上添加一列，用于存放分类 cid 的信息，此列称为：外键





此时“分类表 category”称为：主表，“cid”我们称为主键。“商品表 products”称为：从表，category\_id 称为外键。我们通过主表的主键和从表的外键来描述主外键关系，呈现就是一对多关系。

外键特点：

- ◆ 从表外键的值是对主表主键的引用。
- ◆ 从表外键类型，必须与主表主键类型一致。

- 声明外键约束

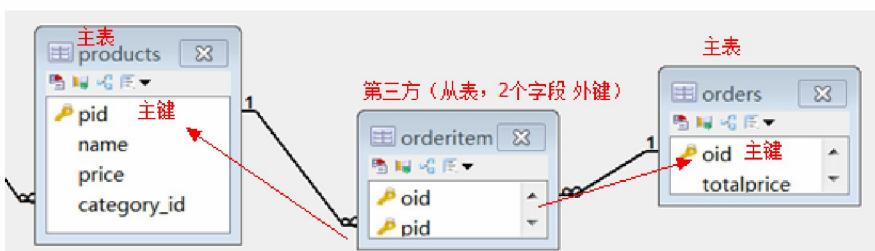
```
语法: alter table 从表 add [constraint] [外键名称] foreign key (从表外键字段名) references
主表 (主表的主键);
[外键名称] 用于删除外键约束的，一般建议“_fk”结尾
alter table 从表 drop foreign key 外键名称
```

- 使用外键目的：

- 保证数据完整性

## 1.3 表与表之间的关系

- 表与表之间的关系，说的就是表与表数据之间的关系。
- 1.一对多关系：
  - 常见实例：客户和订单，分类和商品，部门和员工。
  - 一对多建表原则：在从表(多方)创建一个字段，字段作为外键指向主表(一方)的主键。
- 2.多对多关系：
  - 常见实例：学生和课程，商品和订单，人和角色
  - 多对多关系建表原则：需要创建第三张表，中间表中至少两个字段，这两个字段分别作为外键指向各自一方的主键。（就是将一个多对多拆分成两个一对多。）



- 两张表分别都是主表，第三张表为从表，提供两个字段，都是外键。
- 3.一对一关系：(了解)

- 在实际的开发中应用不多.因为一对多可以创建成一张表.
- 两种建表原则:
  - ◆ 外键唯一: 主表的主键和从表的外键 (唯一), 形成主外键关系, 外键唯一 unique。
  - ◆ 外键是主键: 主表的主键和从表的主键, 形成主外键关系。

## 1.4 创建表实现

### 1.4.1 一对多: 分类和商品

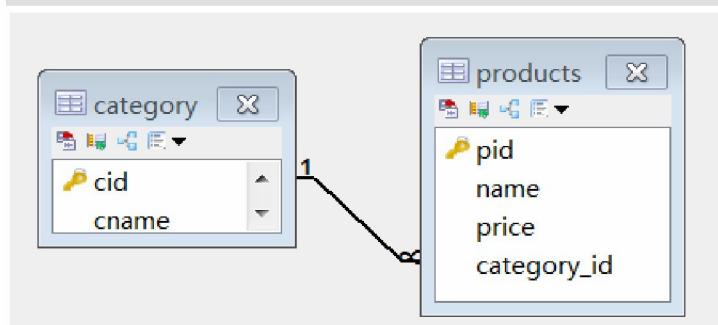
```
#创建数据库
create database day09_db;
#使用数据库
use day09_db;

###创建分类表
create table category(
    cid varchar(32) PRIMARY KEY ,#主表的主键
    cname varchar(100)          #分类名称
);

# 商品表
CREATE TABLE `product` (
    `pid` varchar(32) PRIMARY KEY ,
    `name` VARCHAR(40) ,
    `price` DOUBLE
);

#添加外键字段 category_id
alter table product add column category_id varchar(32);

#添加约束
alter table product add constraint product_fk foreign key (category_id) references
category (cid);
```



- 总结:

- 从表不能够添加（更新），主表中不存在的数据
- 主表不能够删除（更新），从表中已经使用的数据

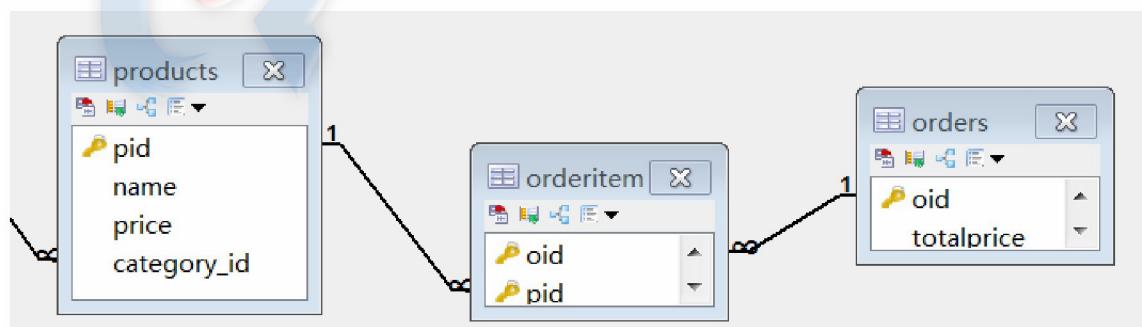
### 1.4.2 多对多：订单和商品

```
#### 订单表
create table `orders`(
    `oid` varchar(32) PRIMARY KEY ,
    `totalprice` double #总计
);

#### 订单项表
create table orderitem(
    oid varchar(50),-- 订单 id
    pid varchar(50)-- 商品 id
);
#### 联合主键（可省略）
alter table `orderitem` add primary key (oid,pid);

####--- 订单表和订单项表的主外键关系
alter table `orderitem` add constraint orderitem_orders_fk foreign key (oid)
references orders(oid);

####--- 商品表和订单项表的主外键关系
alter table `orderitem` add constraint orderitem_product_fk foreign key (pid)
references products(pid);
```



## 1.5 查询操作

### 1.5.1 操作：添加

```
#分类
insert into category(cid,cname) values('c001','家电');
insert into category(cid,cname) values('c002','服饰');
insert into category(cid,cname) values('c003','化妆品');

#商品
insert into product(pid, pname, price, category_id) values('p001', '联想', 5000, 'c001');
insert into product(pid, pname, price, category_id) values('p002', '海尔', 3000, 'c001');
insert into product(pid, pname, price, category_id) values('p003', '雷神', 5000, 'c001');

insert into product (pid, pname, price, category_id) values('p004', 'JACK JONES', 800, 'c002');
insert into product (pid, pname, price, category_id) values('p005', '真维斯', 200, 'c002');
insert into product (pid, pname, price, category_id) values('p006', '花花公子', 440, 'c002');
insert into product (pid, pname, price, category_id) values('p007', '劲霸', 2000, 'c002');

insert into product (pid, pname, price, category_id) values('p008', '香奈儿', 800, 'c003');
insert into product (pid, pname, price, category_id) values('p009', '相宜本草', 200, 'c003');
```

### 1.5.2 操作：查询

#### 1.5.2.1 多表查询

1. 交叉连接查询(基本不会使用-得到的是两个表的乘积)
  - 语法: `select * from A,B;`
2. 内连接查询(使用的关键字 `inner join` -- `inner` 可以省略)
  - 隐式内连接: `select * from A,B where 条件;`
  - 显示内连接: `select * from A inner join B on 条件;`
3. 外连接查询(使用的关键字 `outer join` -- `outer` 可以省略)
  - 左外连接: `left outer join`
    - ◆ `select * from A left outer join B on 条件;`
  - 右外连接: `right outer join`
    - ◆ `select * from A right outer join B on 条件;`

```
#1. 查询哪些分类的商品已经上架
select * from category,products where cid = category_id;
select * from category c,products p where c.cid = p.category_id;
```

```

select cname from category c,products p where c.cid = p.category_id;
#隐式内连接
select distinct cname from category c,products p where c.cid = p.category_id;
#内连接
select distinct cname from category c
inner join products p on c.cid = p.category_id;

```

cname
电器
服饰
化妆品

#2. 查询所有分类上架商品的个数

```

select * from category c
left outer join products p on c.cid = p.category_id;

#左外连接
select cname,count(category_id) from category c
left outer join products p on c.cid = p.category_id group by cname;

```

cname	count(category_id)
书籍	0
化妆品	2
服饰	4
电器	3

### 1.5.2.2 子查询

子查询：一条 select 语句结果作为另一条 select 语法一部分（查询条件，查询结果，表等）。

```

#3 子查询，查询“化妆品”分类上架商品详情
#隐式内连接
select p.* from products p, category c where p.category_id = c.cid and c.cname = '化妆品';
#子查询
select * from products where category_id = (select cid from category where cname = '化妆品');

```

pid	name	price	category_id
p008	香奈儿	800	c003
p009	相宜本草	200	c003

### 1.5.3 注意事项

- 从表外键不是添加，主表中不存在的记录。

- 主表不能删除，从表中已经引用的记录。

## 第2章 SQLyog 安装与使用

- 参考：资料/《SQLyog 安装》

## 第3章 使用JDBC完成分类表CRUD的操作

### 3.1 案例分析

昨天我们已经回顾了 JDBC 操作，今天我们将在此基础上，使用 JDBC 对分类表 category 进行增删改查操作。

### 3.2 工具类

通过上面查询案例进行回顾，我们一起思考一下，发现“获得连接”和“释放资源”两次代码将在之后的增删改查所有功能中都存在，开发中一般遇到此种情况，将采用工具类的方法进行抽取，从而达到代码的重复利用。

cn.itcast.c\_jdbc\_utils\_v1  
JdbcUtils.java

此处提供 v1 版本，之后还有其他版本。

#### 3.2.1 获得连接

```
/**  
 * 获得连接  
 * @return  
 */
```

```

public static Connection getConnection() {
    try {
        //1 注册驱动
        Class.forName("com.mysql.jdbc.Driver");
        //2 获得连接
        Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/day07_db", "root", "1234");
        return conn;
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

```

### 3.2.2 释放资源

如果释放资源采用依次关闭三个对象，那么第一个对象关闭时抛出了异常，后面两个对象将无法成功释放资源，通常我们使用 **try—catch** 块进行处理。

- 方式 1：多个 **try—catch** 块，将资源释放，容易理解。

```

/**
 * 释放资源，多个 try-catch 并列，catch 块中不能抛出异常，否则将阻止程序的继续执行
 * @param conn
 * @param st
 * @param rs
 */
public static void release2(Connection conn, Statement st, ResultSet rs) {
    try {
        if (rs != null) {
            rs.close();
        }
    } catch (Exception e) {
    }

    try {
        if (st != null) {
            st.close();
        }
    } catch (Exception e) {
    }

    try {
        if (conn != null) {
            conn.close();
        }
    }
}

```

```
    } catch (Exception e) {
    }
}
```

● 方式 2：try-catch-finally 嵌套，资源释放时如果出错，将通知调用者

```
/***
 * 释放资源，采用 finally 及时程序抛异常，也可以继续释放其他资源
 * @param conn
 * @param st
 * @param rs
 */
public static void release2(Connection conn, Statement st, ResultSet rs) {
    try {
        if (rs != null) {
            rs.close();
        }
    } catch (Exception e) {
        throw new RuntimeException(e);
    } finally{
        try {
            if (st != null) {
                st.close();
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        } finally{
            try {
                if (conn != null) {
                    conn.close();
                }
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

### 3.2.3 使用 properties 配置文件

开发中获得连接的 4 个参数（驱动、URL、用户名、密码）通常都存在配置文件中，方便后期维护，程序如果需要更换数据库，只需要修改配置文件即可。

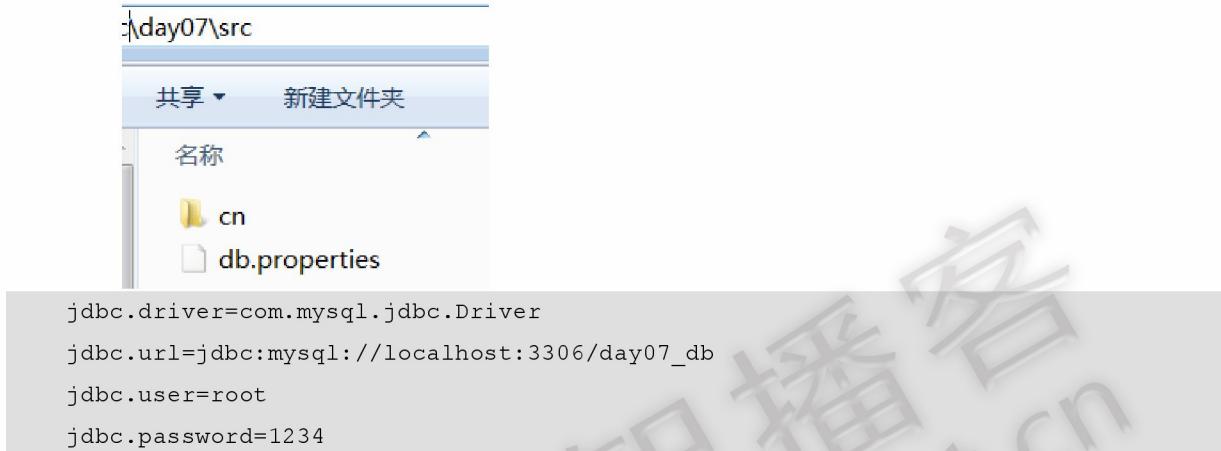
通常情况下，我们习惯使用 **properties** 文件，此文件我们将做如下要求：

1. 文件位置：任意，建议 **src** 下

2. 文件名称：任意，扩展名为 properties
3. 文件内容：一行一组数据，格式是 “key=value” .
  - a) key 命名自定义，如果是多个单词，习惯使用点分隔。例如：jdbc.driver
  - b) value 值不支持中文，如果需要使用非英文字符，将进行 unicode 转换。

### 3.2.3.1 创建配置文件

右键/New/File，输入“db.properties”文件名。



### 3.2.3.2 加载配置文件： ResourceBundle 对象

我们将在 v2 版本中使用 JDK 提供的工具类 ResourceBundle 加载 properties 文件， ResourceBundle 提供 getBundle()方法用于只提供 properties 文件即可，之后使用 getString(key)通过 key 获得 value 的值。

```
public class JdbcUtils {

    private static String driver; //驱动
    private static String url; //路径
    private static String user; //用户名
    private static String password; //密码
    /**
     * 配置文件只需要加载一次，提供静态代码，当前类被加载到内存执行。
     */
    static{
        //1 使用 JDK 提供的工具类加载 properties 文件
        // * ResourceBundle 专门用于处理 properties 文件的，提供 getBundle() 方法只需要填写文件即可
        ResourceBundle bundle = ResourceBundle.getBundle("db");
        //2 通过 key 获得需要的值
    }
}
```

```
        driver = bundle.getString("jdbc.driver");
        url = bundle.getString("jdbc.url");
        user = bundle.getString("jdbc.user");
        password = bundle.getString("jdbc.password");

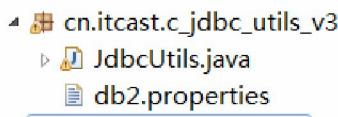
    }
```

### 3.2.3.3 获得连接

```
/**
 * 获得连接
 * @return
 */
public static Connection getConnection() {
    try {
        //1 注册驱动
        Class.forName(driver);
        //2 获得连接
        Connection conn = DriverManager.getConnection(url,user,password);
        return conn;
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

### 3.2.3.4 加载配置文件：Properties 对象（可选）

对应 properties 文件处理，开发中也会使用 Properties 对象进行。在 v3 版本中我们将采用加载 properties 文件获得流，然后使用 Properties 对象进行处理。



在 v3 目录下添加文件 db2.properties，加载文件时对比使用。

```
static{
    try {
        //1 加载 properties 文件，获得流 InputStream
        /* 1.1 方式 1：使用类加载 ClassLoader 加载 src 的资源（固定的写法）
         * * 获得 ClassLoader 固定写法：当前类.class.getClassLoader()
         */
        InputStream is =
JdbcUtils.class.getClassLoader().getResourceAsStream("db.properties");
        /* 1.2 方式 2：加载当前类同包下的资源，如果需要从 src 开始必须填写/
    }
```

```
/*
InputStream is2 = JdbcUtils.class.getResourceAsStream("db2.properties");
//加载的是 cn/itcast/c_jdbc_utils_v3/db2.properties
InputStream is3 = JdbcUtils.class.getResourceAsStream("/db.properties");
//加载的 src 下的资源

//2 使用 Properties 处理流
// * 使用 load() 方法加载指定的流
Properties props = new Properties();
props.load(is);

//2 使用 getProperty(key)，通过 key 获得需要的值，
driver = props.getProperty("jdbc.driver");
url = props.getProperty("jdbc.url");
user = props.getProperty("jdbc.user");
password = props.getProperty("jdbc.password");
} catch (Exception e) {
    throw new RuntimeException(e);
}
}
```

## 3.3 实现

### 3.3.1 模板

```
//模板
@Test
public void demo00() throws Exception{
    Connection conn = null;
    Statement st = null;
    ResultSet rs = null;
    try {
        //1 获得连接
        conn = JdbcUtils.getConnection();

    } catch (Exception e) {
    } finally{
        //释放资源
        JdbcUtils.release(conn, st, rs);
    }
}
```

```
}
```

### 3.3.2 添加：insert into

```
//添加
@Test
public void demo01() throws Exception{
    Connection conn = null;
    Statement st = null;
    ResultSet rs = null;
    try {
        //1 获得连接
        conn = JdbcUtils.getConnection();

        //2 获得语句执行者
        st = conn.createStatement();
        //3 执行语句
        int r = st.executeUpdate("insert into category(cid,cname) values('c001','
家电')");
        //4 输出结果
        System.out.println(r);

    } catch (Exception e) {
    } finally{
        //释放资源
        JdbcUtils.release(conn, st, rs);
    }
}
```

### 3.3.3 更新：update ...set

```
//更新
@Test
public void demo02() throws Exception{
    Connection conn = null;
    Statement st = null;
    ResultSet rs = null;
    try {
        //1 获得连接
        conn = JdbcUtils.getConnection();
```

```
//2 获得语句执行者
st = conn.createStatement();
//3 执行语句
int r = st.executeUpdate("update category set cname='服饰' where cid =
'c001'");
//4 输出结果
System.out.println(r);

} catch (Exception e) {
} finally{
    //释放资源
    JdbcUtils.release(conn, st, rs);
}
}
```

### 3.3.4 删除：delete

```
//删除
@Test
public void demo03() throws Exception{
    Connection conn = null;
    Statement st = null;
    ResultSet rs = null;
    try {
        //1 获得连接
        conn = JdbcUtils.getConnection();

        //2 获得语句执行者
        st = conn.createStatement();
        //3 执行语句
        int r = st.executeUpdate("delete from category where cid = 'c001'");
        //4 输出结果
        System.out.println(r);

    } catch (Exception e) {
    } finally{
        //释放资源
        JdbcUtils.release(conn, st, rs);
    }
}
```

### 3.3.5 通过 ID 查询

```
//通过 id 查询
@Test
public void demo04() throws Exception{
    Connection conn = null;
    Statement st = null;
    ResultSet rs = null;
    try {
        //1 获得连接
        conn = JdbcUtils.getConnection();

        //2 获得语句执行者
        st = conn.createStatement();

        //3 执行语句，获得结果集对象
        rs = st.executeQuery("select * from category where cid='c003'");

        //4 输出结果
        if(rs.next()){
            String cid = rs.getString("cid"); //通过【列名】获得值
            String cname = rs.getString(2); //通过【索引号】获得值，从 1 开始

            System.out.println(cid + " : " + cname);
        } else {
            System.out.println("没有查询结果");
        }
    } catch (Exception e) {
    } finally{
        //释放资源
        JdbcUtils.release(conn, st, rs);
    }
}
```

## 第4章 总结

