

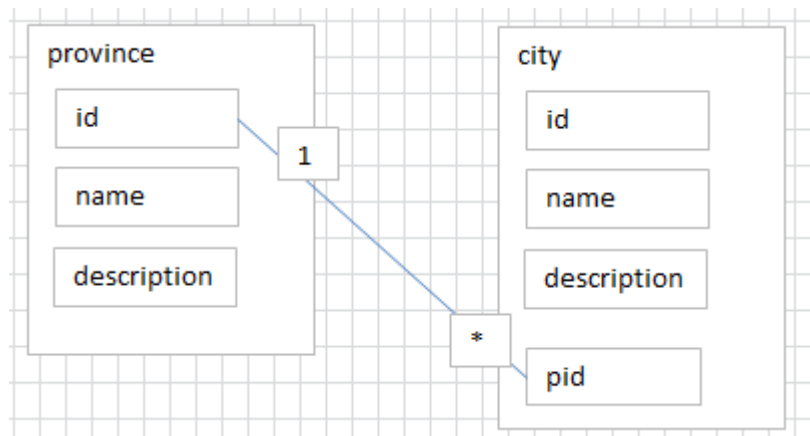
mysql多表查询&原生jdbc

- ☐ 能够使用内连接进行多表查询
- ☐ 能够使用外连接进行多表查询
- ☐ 能够使用子查询进行多表查询
- ☐ 能够理解JDBC的概念
- ☐ 能够使用DriverManager类
- ☐ 能够使用Connection接口
- ☐ 能够使用Statement接口
- ☐ 能够使用ResultSet接口

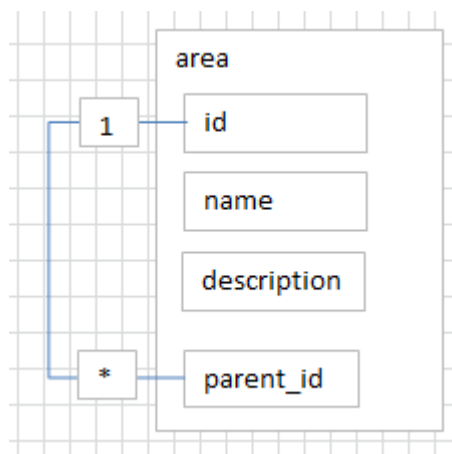
第一章 多表关系实战

1.1 实战1：省和市

- 方案1：多张表，一对多

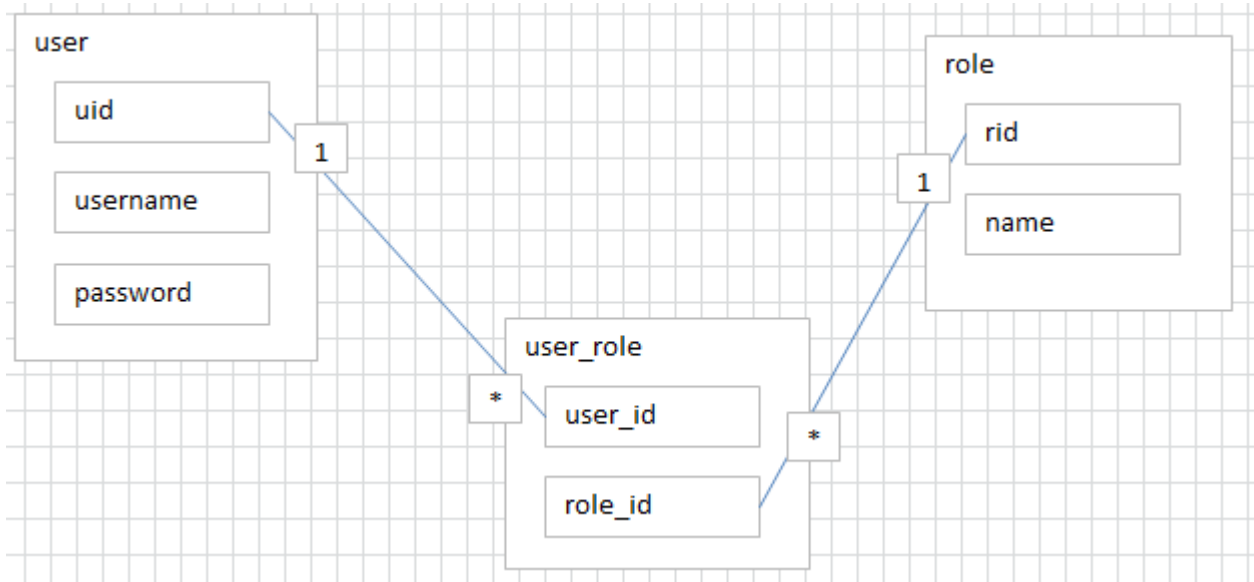


- 方案2：一张表，自关联一对多



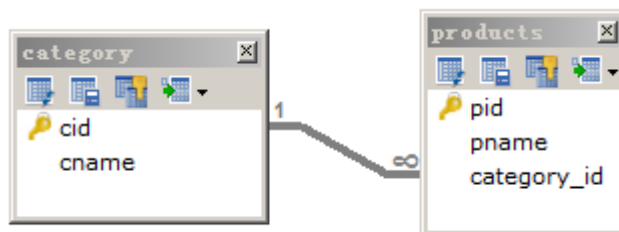
1.2 实战2：用户和角色

- 多对多关系



第二章 多表查询

提供表结构如下：



分类表

```
CREATE TABLE category (  
  cid VARCHAR(32) PRIMARY KEY ,  
  cname VARCHAR(50)  
);
```

#商品表

```
CREATE TABLE products(  
  pid VARCHAR(32) PRIMARY KEY ,  
  pname VARCHAR(50),  
  price INT,  
  flag VARCHAR(2), #是否上架标记为：1表示上架、0表示下架  
  category_id VARCHAR(32),  
  CONSTRAINT products_fk FOREIGN KEY (category_id) REFERENCES category (cid)  
);
```

2.1 初始化数据

#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 products(pid, pname,price,flag,category_id) VALUES('p001','联想',5000,'1','c001');
INSERT INTO products(pid, pname,price,flag,category_id) VALUES('p002','海尔',3000,'1','c001');
INSERT INTO products(pid, pname,price,flag,category_id) VALUES('p003','雷神',5000,'1','c001');

INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p004','JACK JONES',800,'1','c002');
INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p005','真维斯',200,'1','c002');
INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p006','花花公子',440,'1','c002');
INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p007','劲霸',2000,'1','c002');

INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p008','香奈儿',800,'1','c003');
INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p009','相宜本草',200,'1','c003');
```

2.2 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) [了解]

- 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;`

#1. 查询哪些分类的商品已经上架

#隐式内连接

```
SELECT DISTINCT c.cname
FROM category c , products p
WHERE c.cid = p.category_id AND p.flag = '1';
```

#内连接

```
SELECT DISTINCT c.cname
FROM category c INNER JOIN products p
ON c.cid = p.category_id
WHERE p.flag = '1';
```

	cname
<input type="checkbox"/>	电器
<input type="checkbox"/>	服饰
<input type="checkbox"/>	化妆品

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;`

#2. 查询所有分类商品的个数

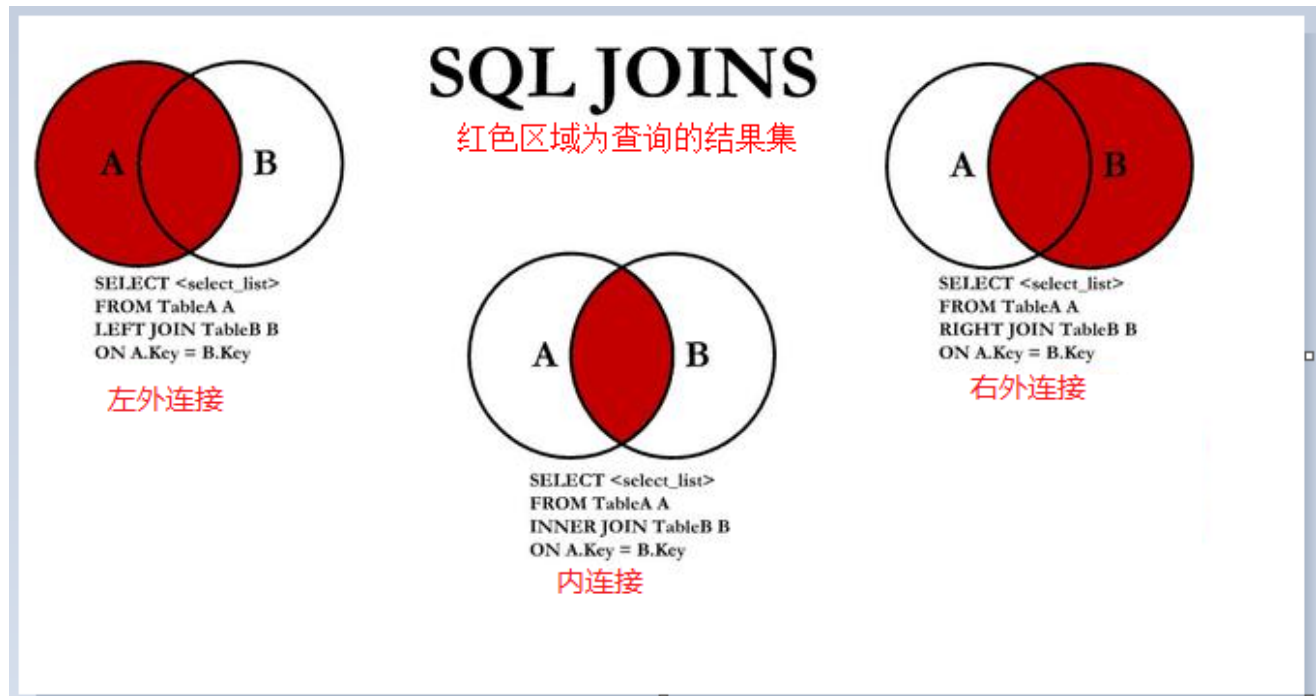
#左外连接

```
INSERT INTO category(cid,cname) VALUES('c004','奢侈品');
```

```
SELECT cname,COUNT(category_id)
FROM category c LEFT OUTER JOIN products p
ON c.cid = p.category_id
GROUP BY cname;
```

	cname	count (p.category_id)
<input type="checkbox"/>	化妆品	2
<input type="checkbox"/>	奢侈品	0
<input type="checkbox"/>	家电	3
<input type="checkbox"/>	服饰	4

下面通过一张图说明连接的区别:



2.3 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）。**语法**：`select`

查询字段 ... from ... 表.. where ... 查询条件

#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';
```

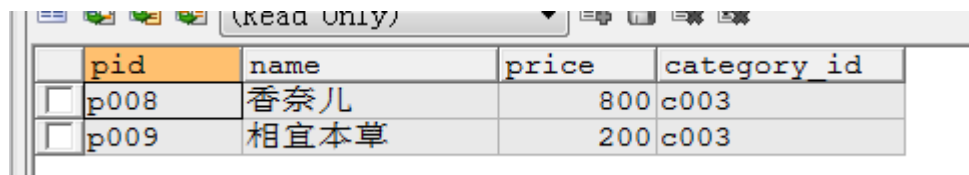
#子查询

##作为查询条件

```
SELECT *  
FROM products p  
WHERE p.category_id =  
    (  
        SELECT c.cid FROM category c  
        WHERE c.cname='化妆品'  
    );
```

##作为另一张表

```
SELECT *  
FROM products p ,  
    (SELECT * FROM category WHERE cname='化妆品') c  
WHERE p.category_id = c.cid;
```



The screenshot shows a database table with the following data:

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

子查询练习：

#查询“化妆品”和“家电”两个分类上架商品详情

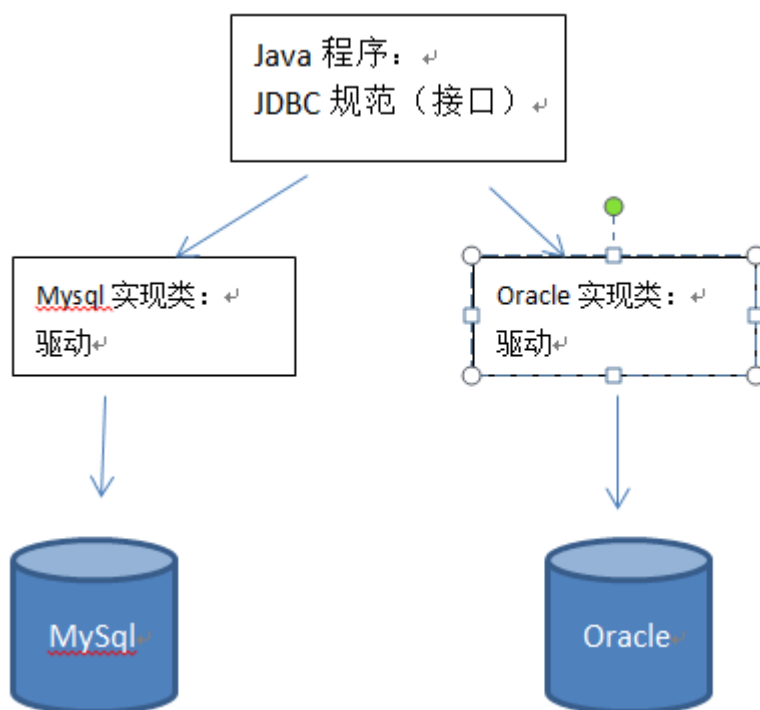
```
SELECT *  
FROM products p  
WHERE p.category_id in  
    (  
        SELECT c.cid  
        FROM category c  
        WHERE c.cname='化妆品' or c.name='家电'  
    );
```

第三章 JDBC

3.1 JDBC概述

JDBC (Java DataBase Connectivity, java数据库连接) 是一种用于执行SQL语句的Java API。JDBC是Java访问数据库的标准规范，可以为不同的关系型数据库提供统一访问，它由一组用Java语言编写的接口和类组成。

JDBC需要连接驱动，驱动是两个设备要进行通信，满足一定通信数据格式，数据格式由设备提供商规定，设备提供商为设备提供驱动软件，通过软件可以与该设备进行通信。今天我们使用的是mysql的驱动mysql-connector-java-5.1.37-bin.jar

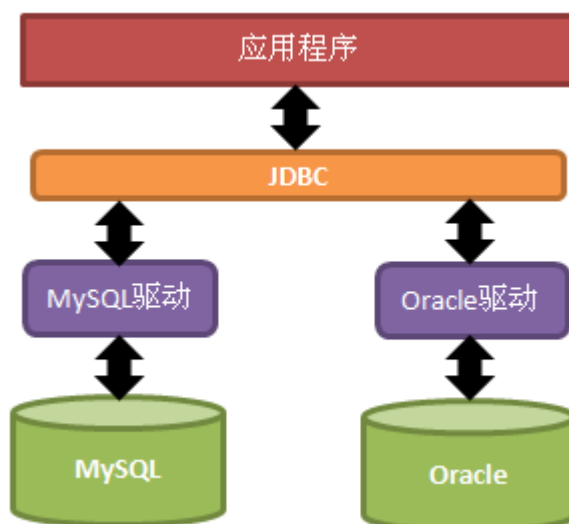


JDBC规范（掌握四个核心对象）：

- DriverManager: 用于注册驱动
- Connection: 表示与数据库创建的连接
- Statement: 操作数据库sql语句的对象
- ResultSet: 结果集或一张虚拟表

3.2 JDBC原理

Java提供访问数据库规范称为JDBC，而生产厂商提供规范的实现类称为驱动。



JDBC是接口，驱动是接口的实现，没有驱动将无法完成数据库连接，从而不能操作数据库！每个数据库厂商都需要提供自己的驱动，用来连接自己公司的数据库，也就是说驱动一般都由数据库生成厂商提供。

3.3 JDBC入门案例

准备数据

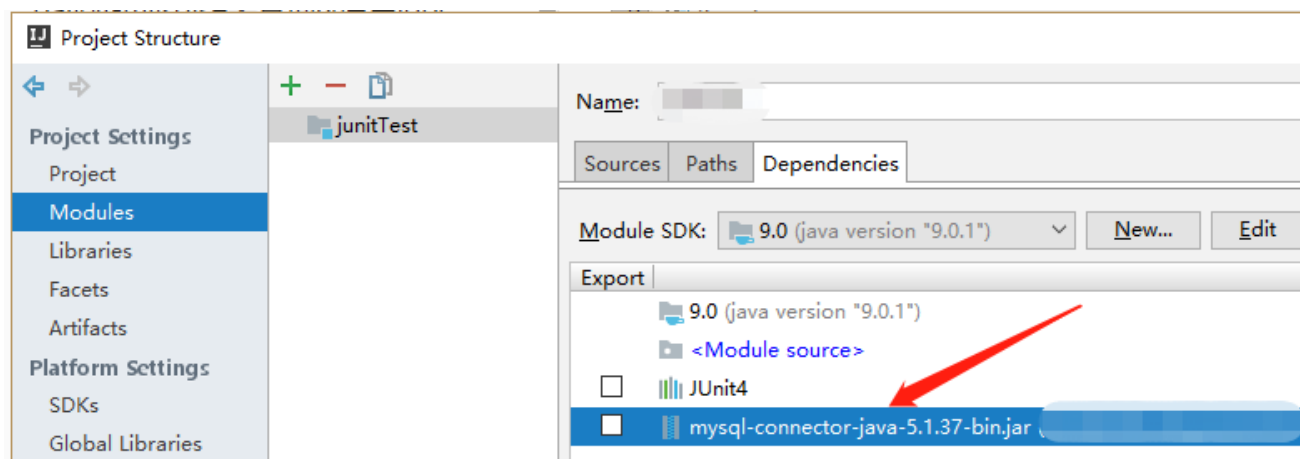
之前我们学习了sql语句的使用，并创建的分类表category，今天我们将使用JDBC对分类表进行增删改查操作。

```
#创建数据库
create database day04;
#使用数据库
use day04;
#创建分类表
create table category(
    cid int PRIMARY KEY AUTO_INCREMENT ,
    cname varchar(100)
);
#初始化数据
insert into category (cname) values('家电');
insert into category (cname) values('服饰');
insert into category (cname) values('化妆品');
```

导入驱动jar包

创建lib目录，存放mysql的驱动mysql-connector-java-5.1.37-bin.jar

通过Project Structure 完成导入



开发步骤

1. 注册驱动.
2. 获得连接.
3. 获得执行sql语句的对象
4. 执行sql语句，并返回结果
5. 处理结果
6. 释放资源.

案例实现

```
//查询所有的分类信息
@Test
public void demo1() throws Exception{
    // 注意：使用JDBC规范，采用都是 java.sql包下的内容
    //1 注册驱动
    Class.forName("com.mysql.jdbc.Driver");
    //2 获得连接
    String url = "jdbc:mysql://localhost:3306/mydb";
    Connection conn = DriverManager.getConnection(url, "root", "root");
    //3获得执行sql语句的对象
    Statement stmt = conn.createStatement();
    //4执行SQL语句
    ResultSet rs = stmt.executeQuery("select * from category");

    //5处理结果集
    while(rs.next()){
        // 获得一行数据
        Integer cid = rs.getInt("cid");
        String cname = rs.getString("cname");
        System.out.println(cid + " , " + cname);
    }
    //6释放资源
    rs.close();
    stmt.close();
    conn.close();
}
```

3.4 API详解

API详解：注册驱动

`DriverManager.registerDriver(new com.mysql.jdbc.Driver());` 不建议使用，原因有2个：

- 导致驱动被注册2次
- 强烈依赖数据库的驱动jar

解决办法：

- `Class.forName("com.mysql.jdbc.Driver");`

API详解：获得链接

`static Connection getConnection(String url, String user, String password)` :试图建立到给定数据库 URL 的连接。

- 参数说明：
 - url 需要连接数据库的位置（网址）
 - user用户名
 - password 密码

- 例如：`getConnection("jdbc:mysql://localhost:3306/day04", "root", "root");`

扩展：

URL:SUN公司与数据库厂商之间的一种协议。

`jdbc:mysql://localhost:3306/day04`

协议子协议 IP :端口号数据库 `mysql: jdbc:mysql://localhost:3306/day04` 或者 `jdbc:mysql:///day04` (默认本机连接) `oracle数据库: jdbc:oracle:thin:@localhost:1521:sid`

API详解：java.sql.Connection接口：一个连接

接口的实现在数据库驱动中。所有与数据库交互都是基于连接对象的。

- `Statement createStatement();` //创建操作sql语句的对象

API详解：java.sql.Statement接口: 操作sql语句，并返回相应结果

```
String sql = "某SQL语句";  
获取Statement语句执行平台：Statement stmt = con.createStatement();
```

常用方法：

- `int executeUpdate(String sql);` --执行insert update delete语句.
- `ResultSet executeQuery(String sql);` --执行select语句.
- `boolean execute(String sql);` --仅当执行select并且有结果时才返回true，执行其他的语句返回false.

API详解：处理结果集（注：执行insert、update、delete无需处理）

ResultSet实际上就是一张二维的表格，我们可以调用其 `boolean next()` 方法指向某行记录，当第一次调用 `next()` 方法时，便指向第一行记录的位置，这时就可以使用ResultSet提供的 `getXXX(int col)` 方法来获取指定列的数据：(与数组索引从0开始不同，这里索引从1开始)

```
rs.next(); //指向第一行  
rs.getInt(1); //获取第一行第一列的数据
```

常用方法：

- `Object getObject(int index)` / `Object getObject(String name)` 获得任意对象
- `String getString(int index)` / `String getString(String name)` 获得字符串
- `int getInt(int index)` / `int getInt(String name)` 获得整形
- `double getDouble(int index)` / `double getDouble(String name)` 获得双精度浮点型

API详解：释放资源

与IO流一样，使用后的东西都需要关闭！关闭的顺序是先得到的后关闭，后得到的先关闭。

```
rs.close();
stmt.close();
con.close();
```

3.5 JDBC工具类

“获得数据库连接”操作，将在以后的增删改查所有功能中都存在，可以封装工具类JdbcUtils。提供获取连接对象的方法，从而达到代码的重复利用。

该工具类提供方法：`public static Connection getConnection()`。代码如下：

```
public class JdbcUtils {

    private static String driver = "com.mysql.jdbc.Driver";
    private static String url = "jdbc:mysql://localhost:3306/webdb_4";
    private static String user = "root";
    private static String password = "root";

    static{
        try {
            //注册驱动
            Class.forName(driver);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * 获得连接
     * @return
     * @throws SQLException
     */
    public static Connection getConnection() throws SQLException{
        //获得连接
        Connection conn = DriverManager.getConnection(url, user, password);
        return conn;
    }

    /**
     * 释放资源
     * @param conn
     * @param st
     * @param rs
     */
    public static void closeResource(Connection conn , Statement st , ResultSet rs){

        if(rs != null){
            try {
                rs.close();
            } catch (SQLException e) {
```

```

    }
}

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

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

```

3.6 JDBC增删改查操作

插入

```

@Test
public void demo01(){
    //添加

    Connection conn = null;
    Statement st = null;
    ResultSet rs = null;

    try {
        //1 获得连接
        conn = JdbcUtils.getConnection();

        //操作
        //1) 获得语句执行者
        st = conn.createStatement();
        //2) 执行sql语句
        int r = st.executeUpdate("insert into category(cname) values('测试')");

        //3) 处理结果
        System.out.println(r);

    } catch (Exception e) {
        throw new RuntimeException(e);
    } finally{

        //释放资源
    }
}

```

```
        JdbcUtils.closeResource(conn, st, rs);
    }
}
```

修改

```
@Test
public void demo02(){
    //修改
    Connection conn = null;
    Statement st = null;
    ResultSet rs = null;

    try {
        conn = JdbcUtils.getConnection();

        st = conn.createStatement();
        int r = st.executeUpdate("update category set cname='测试2' where cid = 4");
        System.out.println(r);

    } catch (Exception e) {
        throw new RuntimeException(e);
    } finally{
        JdbcUtils.closeResource(conn, st, rs);
    }
}
```

删除

```
@Test
public void demo03(){
    //删除
    Connection conn = null;
    Statement st = null;
    ResultSet rs = null;

    try {
        conn = JdbcUtils.getConnection();

        //操作
        st = conn.createStatement();
        int r = st.executeUpdate("delete from category where cid = 4");
        System.out.println(r);

    } catch (Exception e) {
        throw new RuntimeException(e);
    } finally{
        JdbcUtils.closeResource(conn, st, rs);
    }
}
```

```
}
```

通过id查询详情

```
@Test
public void demo04(){
    //通过id查询详情
    Connection conn = null;
    Statement st = null;
    ResultSet rs = null;

    try {
        conn = JdbcUtils.getConnection();

        //操作
        st = conn.createStatement();
        rs = st.executeQuery("select * from category where cid = 30");

        if(rs.next()){
            String cid = rs.getString("cid");
            String cname = rs.getString("cname");
            System.out.println(cid + " @ " + cname );
        } else {
            System.out.println("没有数据");
        }

    } catch (Exception e) {
        throw new RuntimeException(e);
    } finally{
        JdbcUtils.closeResource(conn, st, rs);
    }
}
```

查询所有

```
@Test
public void demo05(){
    //查询所有
    Connection conn = null;
    Statement st = null;
    ResultSet rs = null;

    try {
        conn = JdbcUtils.getConnection();

        //操作
        st = conn.createStatement();
        rs = st.executeQuery("select * from category");

        while(rs.next()){
            String cid = rs.getString("cid");
```

```
        String cname = rs.getString("cname");
        System.out.println(cid + " @ " + cname );
    }

} catch (Exception e) {
    throw new RuntimeException(e);
} finally{
    JdbcUtils.closeResource(conn, st, rs);
}
}
```