1.多表关系

我们在学习数据库的时候，了解到数据库中多表之前是存在关系的，而这种关系也是固定的，分为:一对多，多对一，一对一和多对多。那么学习完JDBC,JSP,Servlet后。如何在WEB项目中操作数据库呢？这就是我们今天研究的重点:如何使用JAVA代码实现多表关系操作。

2.多表操作之一对多

2.1 数据表

比较经典的一对多的关系就是学生表与年级表，两张表中，学生是多方，年级是一方。因为:一个年级可以有多名学生，但反过来一名学生只属于一个年级。先创建数据表

```
create table student (
  stuid int primary key,
  stuname varchar(5),
  stuage  int,
  gid   int
);
create table grade(
 gradeid int primary key ,
 gname varchar(5)
);
insert into grade values(1,'一年级');
insert into grade values(2,'二年级');
insert into grade values(3,'三年级');
insert into student values(1,'张三',18,1);
insert into student values(2,'李四',14,2);
insert into student values(3,'富贵',13,3);
insert into student values(4,'王芳',17,1);
insert into student values(5,'甜甜',15,2);
```

2.2 创建实体类

要求：类名=表名，列名=属性名(外键列也添加属性)

Student:

```
public class Student {
    private int stuid;
    private String stuName;
    private int stuAge;
    private int gid;
     @Override
    public String toString() {
        return "Student{" +
                "stuid=" + stuid +

                ", stuName='" + stuName + '\'' +
                ", stuAge=" + stuAge +
                ", gid=" + gid +
                '}';
    }

    public int getStuid() {
        return stuid;
    }

    public void setStuid(int stuid) {
        this.stuid = stuid;
    }

    public String getStuName() {
        return stuName;
    }

    public void setStuName(String stuName) {
        this.stuName = stuName;
    }

    public int getStuAge() {
        return stuAge;
    }

    public void setStuAge(int stuAge) {
        this.stuAge = stuAge;
    }

    public int getGid() {
        return gid;
    }

    public void setGid(int gid) {
        this.gid = gid;
    }
}
```

Grade:

```
public class Grade {
    private int gradeId;
    private String gname;

    public int getGradeId() {
        return gradeId;
    }

    public void setGradeId(int gradeId) {
        this.gradeId = gradeId;
    }
    public String getGname() {
        return gname;
    }

    public void setGname(String gname) {
        this.gname = gname;
    }
}
```

2.3 建立两表之间的属性关系

数据表是通过外键列来维系两表关系。实体类是通过属性来维系两表关系。在建立一对多关系时，我们分析到年级是一方，学生是多方。一对多，是以一方为主，所以我们在一方添加多方的一个属性。那这个属性是对象还是集合呢？这里记住一句话:一方存多方的集合，多方存一方的对象。所以需要在年级表中添加下列属性：

Grade新增代码:

```
private List<Student> studentList;
public List<Student> getStudentList() {
    return studentList;
}

public void setStudentList(List<Student> studentList) {
    this.studentList = studentList;
}
```

2.4 创建Dao层接口代码和实现类，操作数据库

Dao层

```
public interface GradeDao {
    //查询某个年级信息(要求:展示年级名称和学生列表)
    public Grade getGradeById(int id);
}
```

实现类:在实现类中需要连接数据库，并且查询结果来自于多张表。此时如何存储数据呢？给大家一个思路:1.在不考虑两表的情况下，先存储各自表中的数据 2.结合上面步骤中添加属性的问题，考虑应该把哪个类添加到另外一个类的属性中。代码如下：

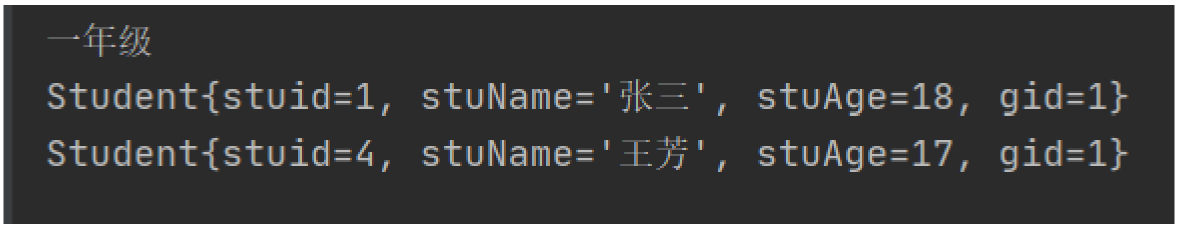
```
public class GradeDaoImpl extends DruidUtil implements GradeDao {
    @Override
    public Grade getGradeById(int id){
        //这里创建年级对象的操作要放在循环外，因为只需要创建一个年级对象即可
        Grade grade = new Grade();
        List<Student> students=new ArrayList<Student>();
        Connection connection =null;
        PreparedStatement preparedStatement =null;
        ResultSet resultSet =null;
        try {
             connection = getConnection();
             preparedStatement = connection.prepareStatement("select * from grade g,student s
where s.gid=g.gradeid and g.gradeid=?");
             preparedStatement.setInt(1,id);
             resultSet = preparedStatement.executeQuery();
            //此时结果集中包含两张表的数据，我们先分别获取各自表中的数据
            while(resultSet.next()){
                //学生信息
                Student student = new Student();
                student.setStuid(resultSet.getInt("stuid"));
                student.setStuName(resultSet.getString("stuname"));
                student.setStuAge(resultSet.getInt("stuage"));
                student.setGid(resultSet.getInt("gid"));
                //年级信息
                grade.setGname(resultSet.getString("gname"));
                grade.setGradeId(resultSet.getInt("gradeid"));
                //建立两者关系
                students.add(student);
            }
            //将学生集合封装到年级中
            grade.setStudentList(students);

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        } finally {
            close(connection,preparedStatement,resultSet);
        }

        return grade;
    }
}
```

说明:这里比较难理解的是关于对象的创建以及属性赋值

2.5 测试类

```
public class Test1 {
    public static void main(String[] args) {

        GradeDao gradeDao = new GradeDaoImpl();
        Grade grade = gradeDao.getGradeById(1);
        System.out.println(grade.getGname());
        List<Student> studentList = grade.getStudentList();
        for (Student student : studentList) {
            System.out.println(student);
        }
    }
}
```

运行结果:



```
一年级
Student{stuid=1, stuName='张三', stuAge=18, gid=1}
Student{stuid=4, stuName='王芳', stuAge=17, gid=1}
```

3.多表操作之多对一

3.1 在上一步的基础上，完成多对一。学生是多方，秉持着"一方存多方的集合，多方存一方的对象"，那么我们就需要在多的一方，添加一方的一个对象。此时学生类中需要添加下列代码

```
private Grade grade;
public Grade getGrade() {
    return grade;
}

public void setGrade(Grade grade) {
    this.grade = grade;
}
```

3.2 在Dao层添加接口方法:

```
public interface StudentDao {
    //查询所有学生的信息(要求包含年级信息)
    public List<Student> getAllStudent();
}
```

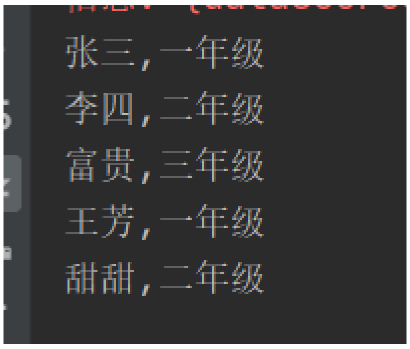
3.3 添加实现类:实现类中主要考虑如何建立两者关联

```
public class StudentDaoImpl extends DruidUtil implements StudentDao {
    @Override
    public List<Student> getAllStudent() {
        //这里创建学生集合对象，放在循环外部
        List<Student> students=new ArrayList<Student>();
        Connection connection =null;
        PreparedStatement preparedStatement =null;
        ResultSet resultSet =null;
        try {
            connection = getConnection();
            preparedStatement = connection.prepareStatement("select * from grade g,student s
where s.gid=g.gradeid ");
            resultSet = preparedStatement.executeQuery();
            //此时结果集中包含两张表的数据，我们先分别获取各自表中的数据
            while(resultSet.next()){
                //学生信息
                Student student = new Student();
                student.setStuid(resultSet.getInt("stuid"));
                student.setStuName(resultSet.getString("stuname"));
                student.setStuAge(resultSet.getInt("stuage"));
                student.setGid(resultSet.getInt("gid"));
                //年级信息
                Grade grade = new Grade();
                grade.setGname(resultSet.getString("gname"));
                grade.setGradeId(resultSet.getInt("gradeid"));
                //建立两者关系
                //将年级封装到学生中
                student.setGrade(grade);
                //将学生封装到学生集合中
                students.add(student);
            }
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        } finally {
            close(connection,preparedStatement,resultSet);
        }

        return students;
    }
}
```

3.4 测试类:

```
public class Test2 {
    public static void main(String[] args) {

       StudentDao studentDao = new StudentDaoImpl();
        List<Student> allStudent = studentDao.getAllStudent();
        for (Student student : allStudent) {
            System.out.println(student.getStuName()+","+student.getGrade().getGname());
        }
    }
}
```

运行结果:



```
张三,一年级
李四,二年级
富贵,三年级
王芳,一年级
甜甜,二年级
```

4.多表操作之一对一

一对一在多表关系中存在场景不是很多，现在以妻子和丈夫的关系，模拟一对一的实现过程。

4.1 创建数据表

```
create table wife(
   wifeid int PRIMARY key,
   wifename varchar(5)
);

create table husband(
 husid int PRIMARY KEY,
 husname varchar(5),
 wid int
);
insert into wife values(1,'黄晓明');
insert into wife values(2,'邓超');

insert into husband values(1,'baby',1);
insert into husband values(2,'孙俪',2);
```

4.2 创建实体类

```
public class Husband {
    private int husId;
    private String husName;
    private int wid;
    //setter and getter
}
```

```
public class Wife {
    private int wifeId;
    private String wifeName;
 //setter and getter
}
```

建立实体类之间的一对一关系，还是依据“一方存多方的集合，多方存一方的对象”的原则，但是现在的问题是双方都是一方数据，此时记住原则“一方存另一方的对象”。所以代码改成：

妻子一方添加丈夫的对象

```
public class Wife {
    private int wifeId;
    private String wifeName;
    private Husband husband;
    //setter and getter
}
```

丈夫一方添加妻子的对象

```
public class Husband {
    private int husId;
    private String husName;
    private int wid;
    private Wife wife;
    //setter and getter
}
```

4.3 添加Dao和实现类

Dao:

```
public interface WifeDao {
    //查询妻子信息(要求包含丈夫信息)
    public Wife getByWifeId(int wifeId);
    //查询丈夫信息(要求包含妻子信息)
    public Husband getByHusId(int husId);
}
```

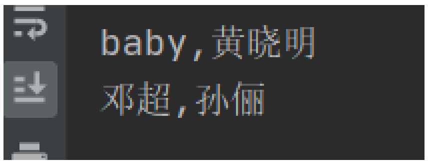
实现类:

```
public class WifeDaoImpl extends DruidUtil implements WifeDao {
    @Override
    public Wife getByWifeId(int wifeId) {

        //这里创建妻子对象
        Wife wife = new Wife();
        Connection connection =null;
        PreparedStatement preparedStatement =null;
        ResultSet resultSet =null;
        try {
            connection = getConnection();
            preparedStatement = connection.prepareStatement("select * from wife w,husband h 
where w.wifeid=h.wid and w.wifeid=? ");
            preparedStatement.setInt(1,wifeId);
            resultSet = preparedStatement.executeQuery();
            //此时结果集中包含两张表的数据，我们先分别获取各自表中的数据
            while(resultSet.next()){
                //妻子信息
                wife.setWifeId(resultSet.getInt("wifeid"));
                wife.setWifeName(resultSet.getString("wifename"));
                //丈夫信息
                Husband husband = new Husband();
                husband.setHusId(resultSet.getInt("husid"));
                husband.setHusName(resultSet.getString("husname"));
                //建立两者关系
                //将丈夫封装到妻子对象中
                wife.setHusband(husband);
            }
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        } finally {
            close(connection,preparedStatement,resultSet);
        }

        return wife;
    }

    @Override
    public Husband getByHusId(int husId) {

        //这里创建丈夫对象
        Husband husband = new Husband();
        Connection connection =null;
        PreparedStatement preparedStatement =null;
        ResultSet resultSet =null;
        try {
            connection = getConnection();
            preparedStatement = connection.prepareStatement("select * from wife w,husband h
where w.wifeid=h.wid and h.husid=?");
            preparedStatement.setInt(1,husId);
            resultSet = preparedStatement.executeQuery();
            //此时结果集中包含两张表的数据，我们先分别获取各自表中的数据
            while(resultSet.next()){
                //妻子信息
                Wife wife = new Wife();
                wife.setWifeId(resultSet.getInt("wifeid"));
                wife.setWifeName(resultSet.getString("wifename"));
                //丈夫信息
                husband.setHusId(resultSet.getInt("husid"));
                husband.setHusName(resultSet.getString("husname"));
                //建立两者关系
                //将妻子封装到丈夫对象中
                husband.setWife(wife);
            }
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        } finally {
            close(connection,preparedStatement,resultSet);
        }
        return husband;
    }
}
```

4.4 测试类

```
public static void main(String[] args) {
    WifeDao wifeDao = new WifeDaoImpl();
    Husband husband = wifeDao.getByHusId(1);
    System.out.println(husband.getHusName()+","+husband.getWife().getWifeName());

    Wife wife = wifeDao.getByWifeId(2);
    System.out.println(wife.getWifeName()+","+wife.getHusband().getHusName());
}
```

运行结果:



```
baby,黄晓明
邓超,孙俪
```

5.多表操作之多对多

多对多在现实场景中也是不很多，比较特殊的就是权限列表的三表关系。菜单表和角色表之间属于多对多。某个功能菜单可以分配给多个角色，某个角色也可以拥有多个菜单，在这个分配过程中就是典型的多对多。在多对多中，表的创建也比较有特点，必须是基于三张表来实现。

5.1 创建数据表

```
create table menu(
 menuid int primary key,
 menuname varchar(10)
);
create table role(
 roleid int primary key,
 rolename varchar(10)
);
create table middle(
  middleid int primary key,
  mid int,
  rid int
);
insert into menu values(1,'用户管理');
insert into menu values(2,'菜单管理');
insert into menu values(3,'角色管理');

insert into role values(1,'超级管理员');
insert into role values(2,'管理员');
insert into role values(3,'总经理');

insert into middle values(1,1,1);
insert into middle values(2,2,1);
insert into middle values(3,3,1);

insert into middle values(4,1,2);
insert into middle values(5,2,2);
insert into middle values(6,1,3);
```

5.2 定义实体类：中间表不需要生成实体类

Menu:

```
public class Menu {
    private int menuId;
    private String menuName;
    //getter and setter
}
```

Role:

```
public class Role {
    private int roleId;
    private String roleName;
    //getter and setter
}
```

建立实体类之间的多对多关系，还是依据“一方存多方的集合，多方存一方的对象”的原则，但是现在的问题是双方都是多方数据，此时记住原则“多方存另一方的集合”。代码如下:

Menu:

```
public class Menu {
    private int menuId;
    private String menuName;
    private List<Role> roleList;
     //getter and setter
}
```

Role:

```
public class Role {
    private int roleId;
    private String roleName;
    private List<Menu> menuList;
     //getter and setter
}
```

5.3 定义接口和实现类

Dao:

```java
public interface RoleDao {
    //查询某个角色信息(要求包含角色对应的菜单列表)
    public Role findByRoleId(int roleId);
    //查询某个菜单信息(要求包含菜单对应的角色列表)
    public Menu findByMenuId(int menuId);
}
```

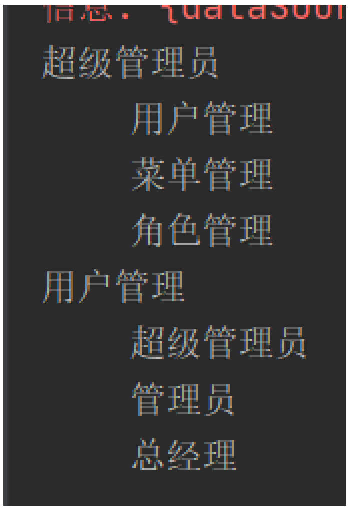
实现类:

```java
public class RoleDaoImpl extends DruidUtil implements RoleDao {
    @Override
    public Role findByRoleId(int roleId) {
        //这里创建角色对象和菜单集合对象
        Role role = new Role();
        List<Menu> menus = new ArrayList<>();
        Connection connection =null;
        PreparedStatement preparedStatement =null;
        ResultSet resultSet =null;
        try {
            connection = getConnection();
            preparedStatement = connection.prepareStatement("select * from role r,menu m,middle
mid where r.roleid=mid.rid and m.menuid= mid.mid and r.roleid=?");
            preparedStatement.setInt(1,roleId);
            resultSet = preparedStatement.executeQuery();
            //此时结果集中包含两张表的数据，我们先分别获取各自表中的数据
            while(resultSet.next()){
                //角色信息
                role.setRoleId(resultSet.getInt("roleid"));
                role.setRoleName(resultSet.getString("rolename"));
                //菜单信息
                Menu menu = new Menu();
                menu.setMenuId(resultSet.getInt("menuid"));
                menu.setMenuName(resultSet.getString("menuname"));
                //建立两者关系
                //将菜单添加到角色的属性中
                menus.add(menu);
            }
            role.setMenuList(menus);
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        } finally {
            close(connection,preparedStatement,resultSet);
        }
        return role;
    }

    @Override
    public Menu findByMenuId(int menuId) {
        //这里创建菜单对象和角色集合对象
        Menu menu = new Menu();
        List<Role> roles = new ArrayList<Role>();
        Connection connection =null;
        PreparedStatement preparedStatement =null;
        ResultSet resultSet =null;
        try {
            connection = getConnection();
            preparedStatement = connection.prepareStatement("select * from role r,menu m,middle
mid where r.roleid=mid.rid and m.menuid= mid.mid and m.menuid=?");
            preparedStatement.setInt(1,menuId);
            resultSet = preparedStatement.executeQuery();
            //此时结果集中包含两张表的数据，我们先分别获取各自表中的数据
            while(resultSet.next()){
                //角色信息
                Role role = new Role();
                role.setRoleId(resultSet.getInt("roleid"));
                role.setRoleName(resultSet.getString("rolename"));
                //菜单信息
                menu.setMenuId(resultSet.getInt("menuid"));
                menu.setMenuName(resultSet.getString("menuname"));
                //建立两者关系
                //将角色添加到菜单的属性中
                roles.add(role);
            }
            menu.setRoleList(roles);
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        } finally {
            close(connection,preparedStatement,resultSet);
        }
        return menu;
    }
}
```

5.4 测试类

```
public static void main(String[] args) {
    RoleDao roleDao = new RoleDaoImpl();
    Role role = roleDao.findByRoleId(1);
    System.out.println(role.getRoleName());
    List<Menu> menuList = role.getMenuList();
    for (Menu menu : menuList) {
        System.out.println("\t"+menu.getMenuName());
    }
    Menu menu = roleDao.findByMenuId(1);
    System.out.println(menu.getMenuName());
    List<Role> roleList = menu.getRoleList();
    for (Role role1 : roleList) {
        System.out.println("\t"+role1.getRoleName());
    }
}
```

运行结果:



```
信息: {dataSour
超级管理员
        用户管理
        菜单管理
        角色管理
用户管理
        超级管理员
        管理员
        总经理
```