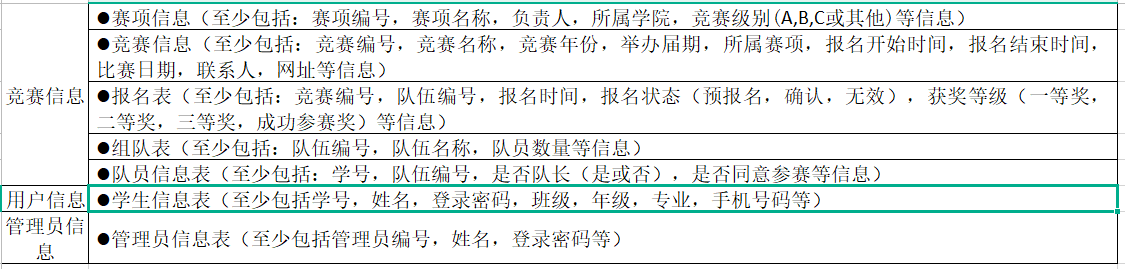
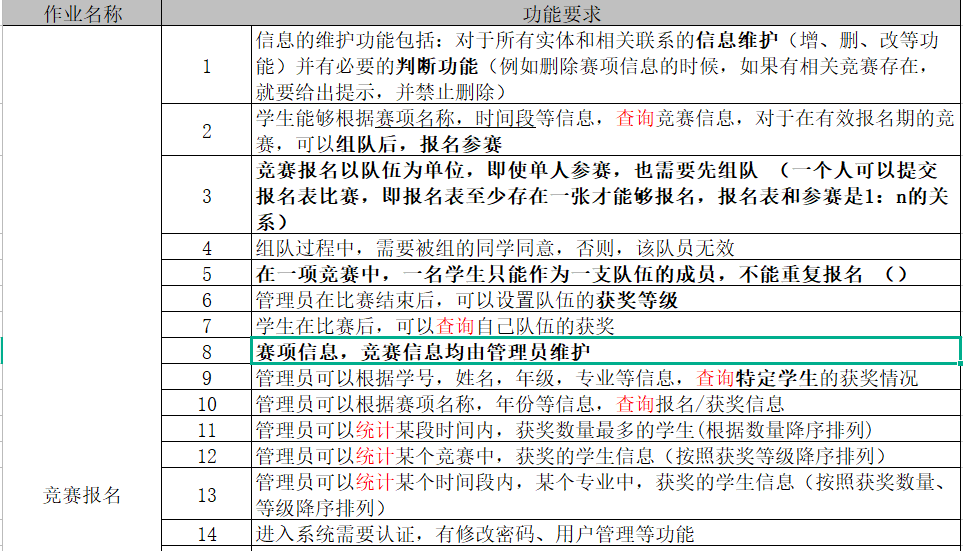
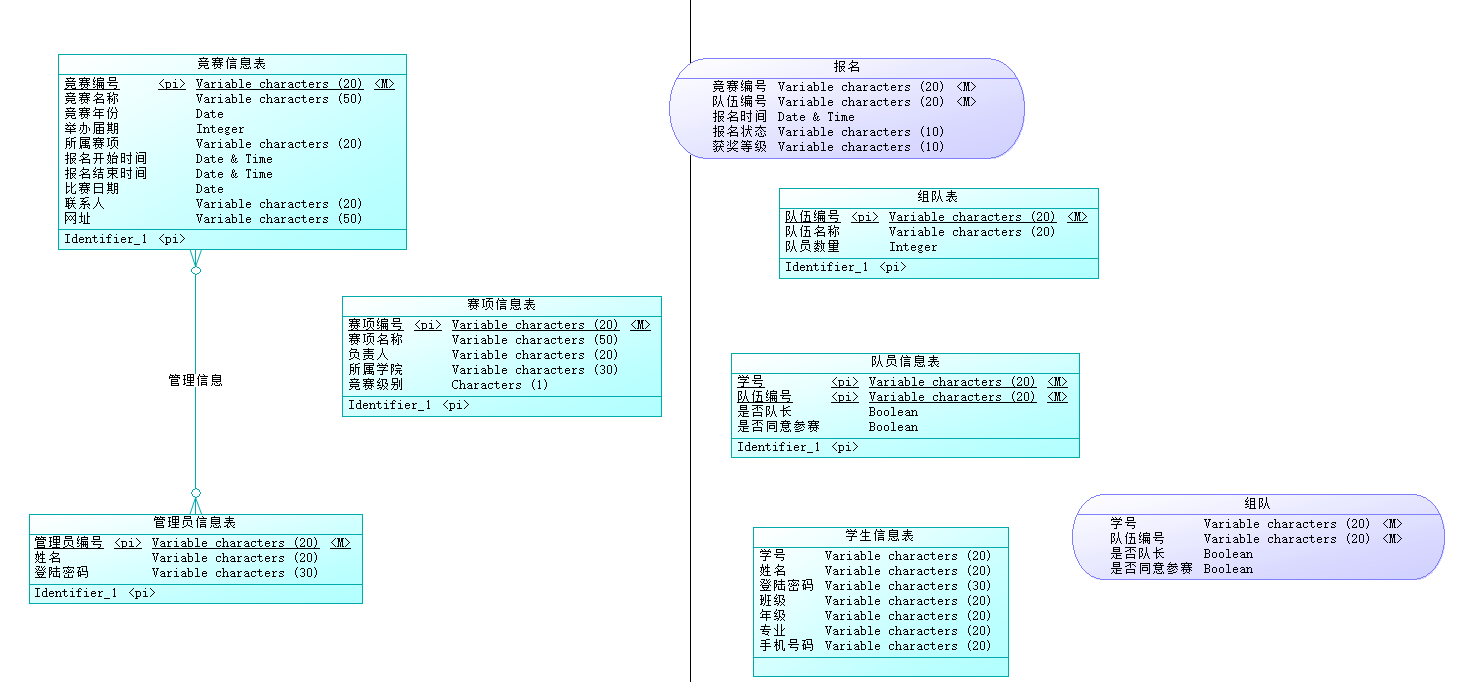
# 竞赛系统概念模型：

### **具体要求如下：**





现设计出的不完善的概念模型为：



现主要问题是表间关系的设计，由设计要求理解的表之间的关系对应准则如下：

准则1：竞赛报名以队伍为单位，即使单人参赛，也需要先组队

准则2：在一项竞赛中，一名学生只能作为一支队伍的成员，不能重复报名

准则3：先组队，后参赛

准则4：赛项信息，竞赛信息均由管理员维护

* + **先组队,后参赛**, 即一个学生必须先成为一个队的队员, 然后才能提交比赛报名表, **即队员信息表相较报名逻辑靠前**
* **学生可以组队成为队员，且不能重复报名,** 即一个学生只能参加一个队伍, 即一个学生只能提交一个组队申请, 即一个学生只能有一个队员信息表, 即学生信息表和队员信息表之间的是一对一的组队关系 (更正如下)

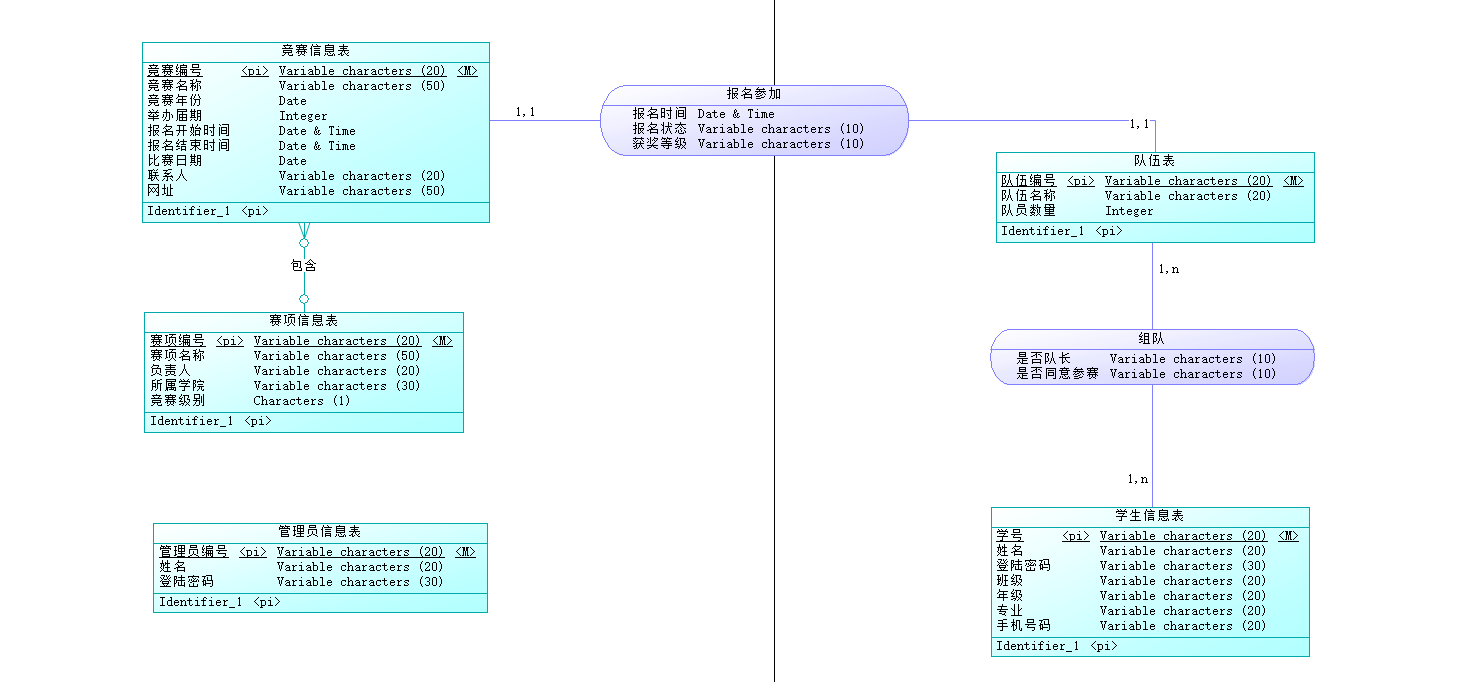
**在一项竞赛中**, 学生只能作为一个队伍的队员, 但是**学生可以参加不同类别的竞赛从而成为多个队伍的队员**, 所以一个学生信息表可以对应多个队员信息表, 一个队员信息表只能对应一个学生信息表, 所以学生信息表和队员信息表应该是一对多的组队关系

* 组队表内是一个队伍的信息, 同时因为队员信息表中含有队伍编号属性, 即一个队员信息表唯一对应一个组队表, 所以队员信息表和组队表之间的关系为一对一
  + 考虑到学生可以参加不同种的竞赛, 即可以提交多个队员信息表, 一个队同时参加多种竞赛的可能性较小, 同时需求中没有提及队伍和竞赛数量之间的对应关系, **所以此处默认一个队伍只参加一个竞赛**, 即组队表和报名之间的关系为一对一, 报名和竞赛信息之间的关系为一对一
  + 一个赛项包含多种竞赛, 一个竞赛属于一种赛项, 即赛项信息和竞赛信息之间的关系为一对多
* 一个管理员可以管理多个竞赛或者赛项的信息, 一个竞赛或者赛项的信息可能被多个管理员进行管理, 所以管理员信息表和赛项信息 竞赛信息之间的关系均为多对多

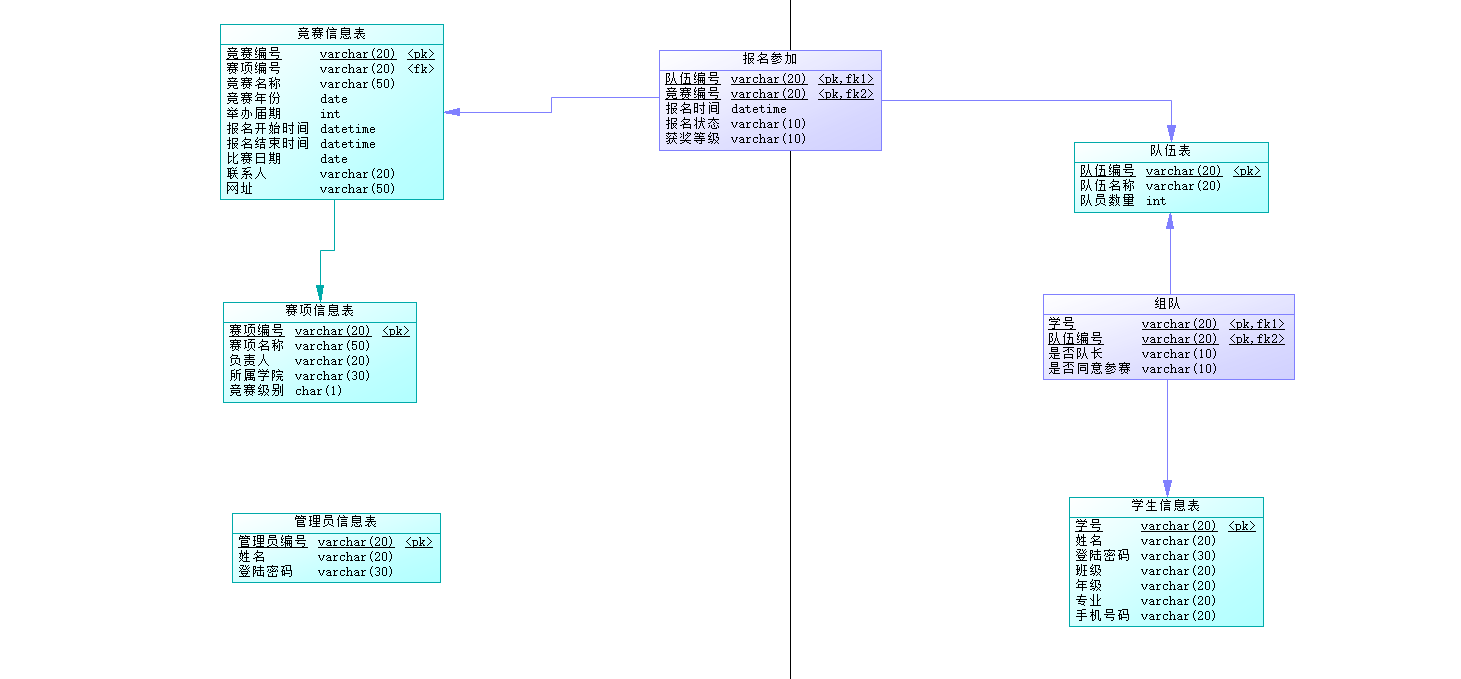
上述条目前标了**两个点**的是正确的:

* 对于学生信息表和队伍表之间的关系:
  + 首先应将组队看作是一个关系, 而非是一个实体, 关系连接两个实体, 在连接时不应包含连接的实体中的属性
  + 在之前我将队伍表和学生信息表理解成了是一对一的关系, 因为我以为一个队伍和一个学生唯一绑定, 但忽略了对于一个队伍中的所有学生来说, 队伍信息都是相同的, 即一个队伍对应多个学生; 与此同时, 一个学生还可以参加不同的队伍, 对于一个学生来说, 可能有着不止一个的队伍信息, 所以队伍表和学生信息表之间的关系应该是多对多.
* 对于管理员管理竞赛或者赛项的关系倒是不必太过在意, 因为没有太过于直接的关联.

完善后的 **CompetitionSystem概念模型**



完善后的 **CompetitionSystem物理模型**



**系统设计：**

cn.zucc.edu.dao **data access object 数据访问对象**

cn.zucc.edu.util **工具包**

cn.zucc.edu.view **界面设计**

cn.zucc.edu.models **实体**

cn.zucc.edu.cn.images **界面设计图标**

**设计过程：**

1. 先创建models包, 有几个表，models包中就有几个类
2. 然后完善view包进行界面设计
3. 最后dao包，将业务代码完善

# view包登录界面设计

## 登录验证

**问题/1\:**

在设计登录时用户名密码匹配验证时, 需要考虑用户有两类, 一类是学生, 一类是管理员,

所以这时不能简单地只将学生的数据库与登录功能进行连接, 管理员的数据库也要进行连接, 即管理员也要创建对象, 然后获取用户名密码信息, 再进行判断

这样会造成代码**较为冗杂**

**解决思路:**

管理员和学生同为用户, 所以可以创建一个用户表, 将学生和管理员的登录信息合并到一起, 从而简化登录时用户名密码判断的复杂程度

但是这样在后期的学生信息查询中可能又会造成一定的麻烦

**问题/2\:**

那在新创建的User表中, 是否要创建一个属性用来区分管理员和学生呢

**解决思路:**

应该是要的, 否则无法区分权限

那如果是这样的话还不如不创建一个新表, 虽然在登录验证的时候代码繁琐了一点, 但是后期在身份区分所导致的权限问题上就简单了许多

这张UserLog表可以先保留, 但是登录验证的时候还是对管理员和用户分别验证登录

这时对于学生来说: 用户名就默认为学号

对于管理员来说: 用户名就默认为管理员ID

**最终思路:**

如果不合成一个表, 在登录判断时要分别判断, 并且之后在进行登录身份查询的时候也要写两个Dao文件, 好处是管理员和学生分的比较开, 在后期权限问题或者查询问题上可能少些麻烦

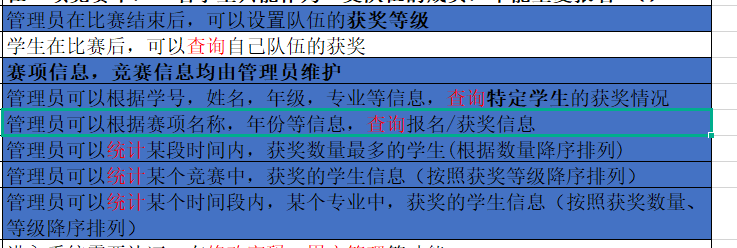
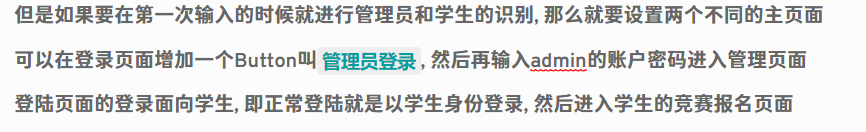
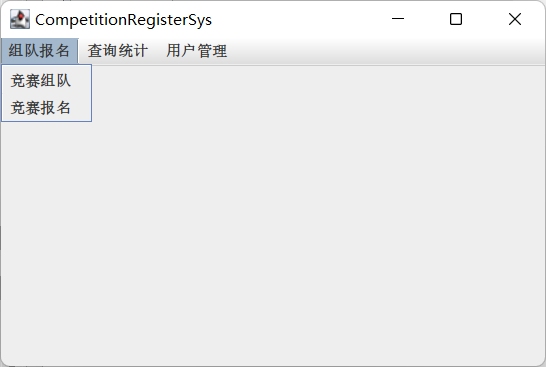
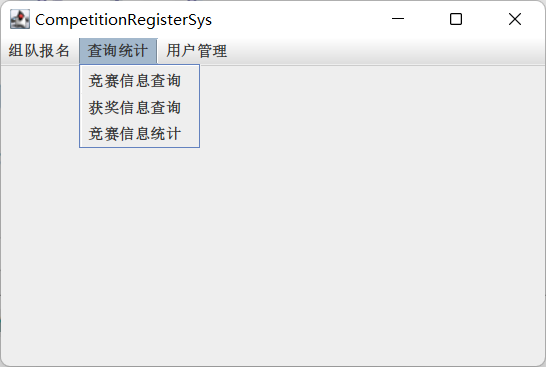
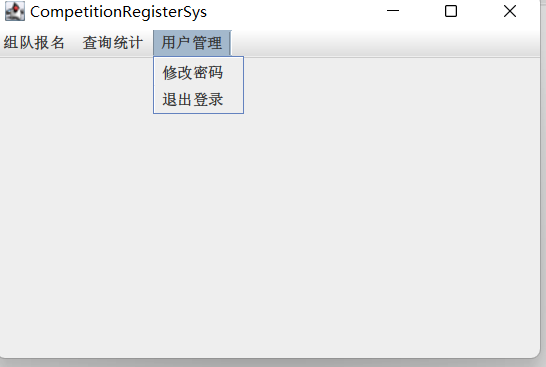
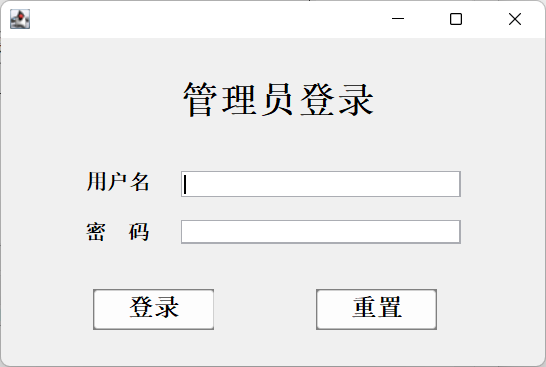
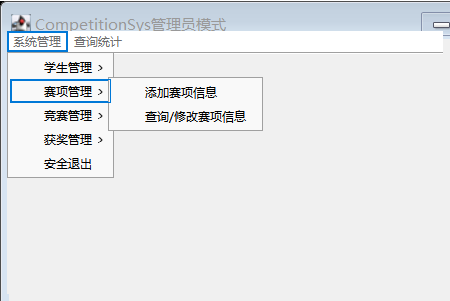
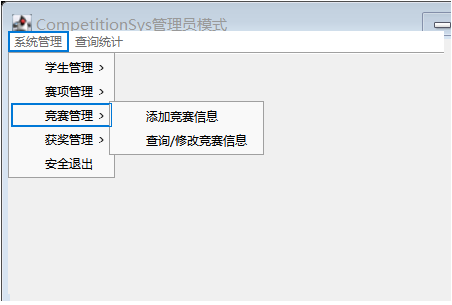
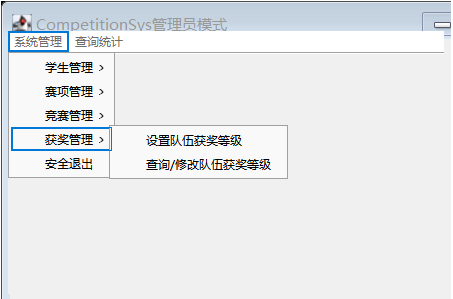
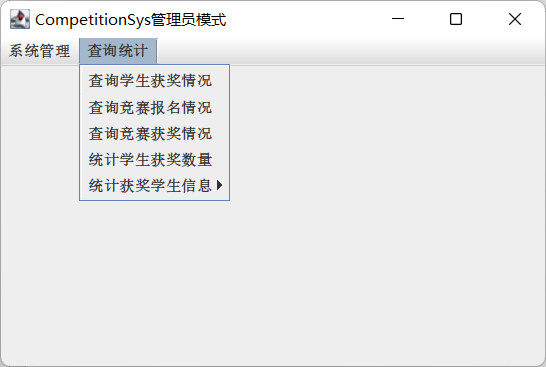
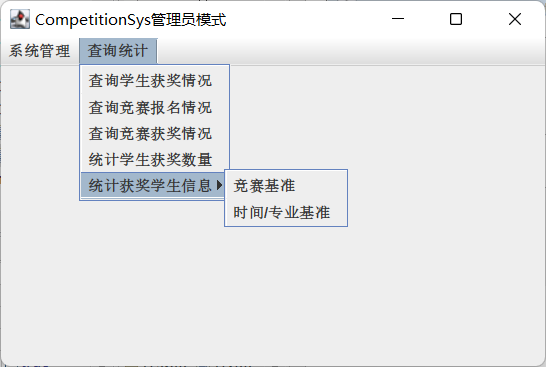
如果合成一个表, 那么在登录判断和登陆身份查询时都可以简单一些, 在此基础上, 可以添加第三个身份属性用来区分管理员和学生, 用来解决权限问题

**所以最终采用多添加一个属性的合成一个UserLog表的方式**

## 页面布局

现在预想的几个问题：

1. 既然采用了统一管理登录的方式, 那么管理员的权限问题应该如何设置
   * **目前的想法是**:

* 在系统中增加一个管理员模式
* 进入管理员模式需要输入管理员ID和密码, 在系统中增加一个二次登陆的方式, 从而区分开学生和管理员
* **管理员的权限**如下:
* 
* **预想页面:**
  + 在主页面上有一个管理员模式
  + 点击管理员模式, 会跳出来一个管理员登录界面
  + 在输入管理员ID和密码后, 会进入一个竞赛信息管维护统计主界面
  + 这样设计其实有点怪, 在第一次登陆的时候管理员已经输入自己的admin帐号了, 进入之后却还要再输入一次, 在使用逻辑上不合理
  + **但是如果要在第一次输入的时候就进行管理员和学生的识别, 那么就要设置两个不同的主页面**
  + **可以在登录页面增加一个Button叫管理员登录, 然后再输入admin的账户密码进入管理页面**
  + **登陆页面的登录面向学生, 即正常登陆就是以学生身份登录, 然后进入学生的竞赛报名页面**
  + 主页面上方依次有赛项管理 竞赛管理 队伍管理 查询统计
  + 赛项管理链接到cptitemsinfo表, 将数据库中的信息展示出来
  + 竞赛管理链接到cptinfo表, 将数据库中的信息展示出来
  + 队伍管理链接到team表, 将数据库中的信息展示出来, 在这里管理员可以设置队伍的获奖等级
  + 对这三个表的管理, 用删除记录/修改记录/添加记录的方式
  + 查询统计里面有查询学生获奖情况 查询竞赛报名情况 查询竞赛获奖情况 统计学生获奖数量 统计获奖学生信息
  + 统计获奖学生信息中分为竞赛基准 时间/专业基准
* **最终页面布局**:
* 采用该种想法:
* 
* **即学生登陆界面作为主界面, 在主界面中添加管理员登陆选项, 用户可以依照自己身份进行选择**
* 
* **若直接登录, 则默认是以学生身份登录, 将出现学生登录界面**
* 学生登陆界面有三个菜单:
  + 一个是组队报名, 其中包括竞赛组队 和 竞赛报名
* 
  + 一个是查询统计, 其中包括竞赛信息查询 获奖信息查询 竞赛信息统计
* 
  + 一个是用户管理, 其中包括修改密码 退出登录
* 
* **若点击管理员选项, 即会弹出管理员登录界面**
* 
* **在管理员登录后会显示管理界面**
* 管理界面有两个菜单:
  + 一个是系统管理, 其中包含学生管理 赛项管理 竞赛管理 获奖管理 安全退出
  + 
    - 四个管理中分为添加 和 查询/修改两个功能板块
    - 
    - 
    - 
    - 
  + 一个是查询统计, 其中包含查询学生获奖情况 查询竞赛报名情况 查询竞赛获奖情况 统计学生获奖数量 , 还有一个分菜单统计获奖学生信息
  + 查询学生获奖情况:
    - 管理员可以根据学号，姓名，年级，专业等信息，查询**特定**学生的获奖情况
  + 查询竞赛报名情况
    - 管理员可以根据赛项名称，年份等信息，查询报名信息
  + 查询竞赛获奖情况
    - 管理员可以根据赛项名称，年份等信息，查询获奖信息
  + 统计学生获奖数量
    - 管理员可以统计某段时间内，获奖数量最多的学生(根据数量降序排列)
  + 统计获奖学生信息有两个查询基准:
    - 一个是以竞赛为基准进行查询, 即统计**某个竞赛中**的获奖学生的信息
    - 一个是以时间和专业为基准进行查询, 即统计**某个时间段内某个专业的**学生获奖信息
* 
* 

现在要做的事情：

* 将学生界面的control部分功能完善
* 在过程中再创建学生页面的UI
* 将学生和管理员的登陆界面连接
* 完善管理员的control功能

现在学生登录界面和管理员登陆界面, 以及学生登陆界面和学生系统主页面、管理员登陆界面和管理员主页面都已经串联完成

弹出窗口有两种:

* 一种是依存于现有窗口弹出
* 一种是**直接弹出**另一个和现有窗口逻辑不关联的新窗口

**核心代码:**

* 依存于现有窗口弹出的
* 在要准备弹出的事件下面添加代码:
* 要弹出的窗口的view类 对象命名 = new 要弹出的窗口的view类(); //创建目的弹出窗口的对象  
   对象命名.setVisible(true);  
   目的弹出窗口从源窗口弹出的位置.add(对象命名); //该位置在本窗口的类中会被系统自动创建和定义,在前面找定义即可
* 直接弹出另一个和现有窗口逻辑不关联的新窗口
* 在要准备弹出的事件下面添加代码:
* /\*\*  
  因为要弹出的窗口和现有窗口没有逻辑上的联系  
  所以不需要再从源窗口的某个位置上弹出,所以add函数可以不用添加  
  只需要创建目的窗口的对象,然后再显示可见即可  
  (默认是不可见)  
  \*/  
  要弹出的窗口的view类 对象命名 = new 要弹出的窗口的view类(); //创建目的弹出窗口的对象  
  对象命名.setVisible(true);

在定义事件考虑了插入变量的数量问题, 因为各个表中的第一个编号属性在使用过程中似乎默认是一个递增变量

而递增的设置需要属性字段为整型

这显然不符合一些表中编号为主码的要求

所以对于编号在填充数据的时候要认为的进行差异化填充

**现定义如下编号填充规则:**

* 学号以3200开头
* 赛项以3000开头
* 竞赛以3100开头
* 队伍编号以3300开头
* 管理员编号从0001开始

基本框架已经全部搭好了, 接下来就是具体到各个核心功能的ui和代码

**现在已经完成的功能:**

登录

重置

退出登录

添加赛项信息

//设置JFrame居中显示  
this.setLocationRelativeTo(null);  
  
//设置最大化  
this.setExtendedState(Frame.MAXIMIZED\_BOTH);

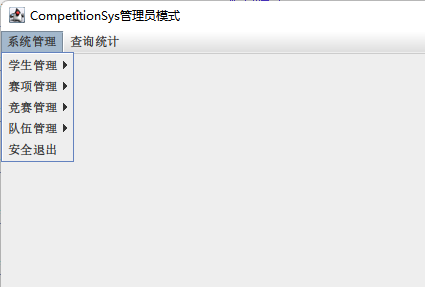
**第七天的任务**：

不要纠结于界面，重点在control

* 先把添加的sql全部弄完，可能在其中有一些判断语句进行，比如判断已存在然后进行提示
* 然后进行查询和修改和删除的sql，集中在一个界面里，这个界面制作的时候参考BookLib的界面和代码
* 问一问老师怎么实现booklib那种弹出式的界面，并且关闭不会连带关闭主界面
* 前两个部分都完成之后，就思考一下学生表和组队表还有报名表的属性分配和相关功能的sql语句

**第七天总结** :

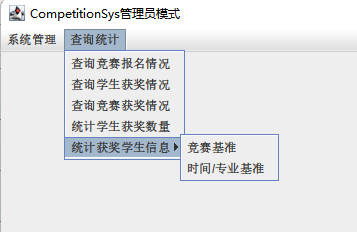
完成竞赛信息 赛项信息 学生信息 组队信息的增删改查



管理员模式中的系统管理模式已经完成

**第八天任务** :

完成查询统计中的内容



添加以下几个view包中的类

AdminQueryCptRegiInterFrm  
AdminQueryStuRwdInterFrm  
AdminQueryCptRwdInterFrm  
AdminCntStuRwdNumInterFrm  
AdminCntRwdStuInfoCptInterFrm  
AdminCntRwdStuInfoTimeInterFrm

这几个类都要好好想一下要怎么定义数据库中的数据和写sql语句, 明天着重完成这几个功能

可以在累的时候把学生修改密码的功能做掉

至此管理员模式的功能全部开发完成

只剩学生模式的功能待开发, 在休息的三天中完成

着重思考进行的模式

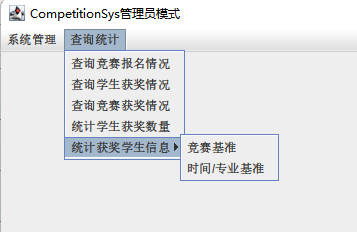
怎么进行竞赛组队和竞赛报名的具体方式, 这两个问题可以在第八天的时候好好想一想, 以便在假期中直接有思路去进行







**第八天的开发过程：**



在构建查询竞赛报名页面的过程中, 考虑到学生报名的方式:

* 是先组队然后队长代表团队报名
* 还是先个人报名, 在报名信息中填写团队编号, 之后在后台再统计相同队伍编号的的学生, 进行队伍的整理.

第二种想法有些复杂, 不易实现, 需要从逻辑上考虑是否符合现实情况

**需要先组队后报名**, 即先要在后台系统中获得已经组队的学生信息, 组队了就代表一定会进行报名, 不然组队干什么(虽然现实会存在组队后放弃报名的情况, 但很少, 暂时不予以考虑)

即学生竞赛组队功能是已经在线下商量好之后组队情况之后的队员填写个人信息

还需要一个队员表, 用来记录参赛队伍中的队员信息, 因为不是所有学生登录系统都要报名参赛成为参赛队员

**以第一种方式为最终方式**:

1. 学生线下组队, 选出队长, 确定自己队伍的队伍编号
2. 每位学生在组队中填写学生编号, 队伍编号, 是否是队长, 是否愿意参赛, 也就是填写team\_up表

* (骚操作, 可能有同学在线下是由于某种原因答应了参赛, 但其实内心是不想参与的, 所以在线下说好回去报名的时候将是否愿意参赛信息改成了不愿意, 最后没有参赛)
* 这时每一个同学都完成了报名的前提条件, 即已 经有了队伍

1. 然后每个队伍的队长去填写报名表

但是不能想当然的就认为学生全部都是线下交流的, 肯定要留有线上直接加入队伍的相关的功能

### 修改密码

/\*\*  
修改密码:  
 原账号:  
 原密码:  
 新密码:  
 确认新密码:  
三个注意点:   
 - 原账号和原密码的确认,即一次登录功能的实现  
 - 新密码和原密码的确认,不能相同  
 - 新密码和确认新密码的确认,不能不同  
同时:所有属性不能为空, 如果有一个空则应提示:请将信息填写完整  
\*/  
StuPswUpdateInterFrm;  
  
//StuDao  
String sql = "update stu\_info set password=? where stuId=?";  
  
//StuPswUpdateInterFrm  
String oldPwd = this.oldPwdTxt.getText();  
String newPwd = this.newPwdTxt.getText();  
String cfrmPwd = this.cfrmPwdTxt.getText();  
  
Connection conn = null;  
if( StringUtil.isEmpty(学号输入) || StrigUtil.isEmpty(老密码输入) ||StringUtil.isEmpty(新密码输入) || StringUtil.isEmpty(确认新密码输入) ){  
 JOptionJane.showMessage(null,"请将信息输入完整");  
}  
try{  
   
}catch(Exception e){  
   
}finally{  
 try{  
   
 }catch(Exception e){  
   
 }  
}  
  
BeanStuInfo stuInfoOld = new BeanStuInfo(stuId,oldPwd);//目的是获取stuId,从而对应数据库中的老密码,以便确认原有登录信息，即是账号持有者进行的修改操作  
BeanStuInfo stuInfoNew =new BeanStuInfo(stuId,newPwd);//目的是获取新设置的密码,从而做新老密码的比较，两个密码不能一样

* **Dao类都要添加异常处理方法**

### 删除信息Delete：

/\*\*  
 \* 竞赛信息删除  
 \* @param evt  
 \*/  
 private void CptInfoDeleteActionPerformed(ActionEvent evt) {  
 String id = textField.getText();  
 if(StringUtil.isEmpty(id)) {  
 JOptionPane.showMessageDialog(null, "请选择要删除的记录");  
 return;  
 }  
 int n = JOptionPane.showConfirmDialog(null, "确认要删除此记录吗？");  
 if(n == 0) {  
 Connection conn = null;  
 try {  
 conn = dbUtil.getConn();  
 int delete = cptDao.delete(conn, id);  
 if(delete == 1) {  
 JOptionPane.showMessageDialog(null, "删除成功");  
 this.reSetValue();//删除之后要清楚之前留在 查询表格 中的信息  
 this.fillTable(new BeanCptInfo());//重新加载 显示表格 中的信息  
 }else {  
 JOptionPane.showMessageDialog(null, "删除失败");  
 }  
 }catch(Exception e) {  
 e.printStackTrace();  
 JOptionPane.showMessageDialog(null, "删除失败");  
 }  
 }  
 }

### 修改信息Update：

1. 对查询表格添加mousePressed事件，将已经显示到查询表格中的信息显示到修改表格中
2. 在修改选项中添加事件
3. 写修改函数

修改函数包含以下要点:

1. 对于空选项的修改报错, 提示请选择要修改的记录
2. 如果信息填写不完整, 提示请将信息填写完整
3. 创建实体对象和连接对象
4. 进行连接, 经过实体对象调用control即Dao data access object数据访问对象中的函数, 对数据库中的数据进行修改. 数据经由面板上的输入, 然后通过getText方法的获取, 从而进行处理
5. 在修改成功之后要清除修改面板中的数据, 同时要在显示面板上显示已经更新过的数据

* 所以要加两个函数, 一个重置修改表单函数resetValue(), 一个填充查询表格函数fillTable()

cptInfoTableMousePressed(e);  
  
 /\*\*  
 \*重置表单   
 \*/  
 private void reSetValue() {  
 this.textField.setText("");  
 this.textField\_1.setText("");  
 this.textField\_2.setText("");  
 this.textField\_3.setText("");  
 this.textField\_4.setText("");  
 this.textField\_5.setText("");  
 this.textField\_6.setText("");  
 this.textField\_7.setText("");  
 this.textField\_8.setText("");  
 this.textField\_9.setText("");  
 }  
  
  
/\*\*  
 \* 表格行点击事件处理   
 \* @param e  
 \*/  
 private void cptInfoTableMousePressed(MouseEvent e) {  
 int row = CptInfoTable.getSelectedRow();  
   
 textField.setText((String)CptInfoTable.getValueAt(row, 0));  
 textField\_1.setText((String)CptInfoTable.getValueAt(row, 1));  
 textField\_2.setText((String)CptInfoTable.getValueAt(row, 2));  
 textField\_3.setText((String)CptInfoTable.getValueAt(row, 3));  
 textField\_4.setText((String)CptInfoTable.getValueAt(row, 4));  
 textField\_5.setText((String)CptInfoTable.getValueAt(row, 5));  
 textField\_6.setText((String)CptInfoTable.getValueAt(row, 6));  
 textField\_7.setText((String)CptInfoTable.getValueAt(row, 7));  
 textField\_8.setText((String)CptInfoTable.getValueAt(row, 8));  
 textField\_9.setText((String)CptInfoTable.getValueAt(row, 9));  
 }  
  
  
CptInfoUpdateActionPerformed(e);  
  
/\*\*  
 \* 竞赛信息修改  
 \* @param e  
 \*/  
 private void CptInfoUpdateActionPerformed(ActionEvent evt) {  
 // TODO 自动生成的方法存根  
 String cptId = textField.getText();  
 String cptItemId = textField\_1.getText();  
 String cptName = textField\_2.getText();  
 String cptYear = textField\_3.getText();  
 String holdYear = textField\_4.getText();  
 String holdDay = textField\_5.getText();  
 String regiStart = textField\_6.getText();  
 String regiEnd = textField\_7.getText();  
 String contact = textField\_8.getText();  
 String web = textField\_9.getText();  
   
 if(StringUtil.isEmpty(cptId)) {  
 JOptionPane.showMessageDialog(null, "请选择要修改的记录");  
 return;  
 }  
 if(StringUtil.isEmpty(cptId) || StringUtil.isEmpty(cptItemId) || StringUtil.isEmpty(cptName)   
 || StringUtil.isEmpty(cptYear) || StringUtil.isEmpty(holdYear) || StringUtil.isEmpty(holdDay)   
 || StringUtil.isEmpty(regiStart) || StringUtil.isEmpty(regiEnd) || StringUtil.isEmpty(contact)   
 || StringUtil.isEmpty(web)){  
 JOptionPane.showMessageDialog(null, "请将要修改的信息填写完整");  
 return;  
 }  
   
 BeanCptInfo cptInfo = new BeanCptInfo(cptId,cptItemId,cptName,cptYear,holdYear,holdDay,regiStart,regiEnd,contact,web);  
 Connection conn = null;  
   
 try {  
 conn = dbUtil.getConn();  
 int rs = cptDao.update(conn, cptInfo);  
 if(rs == 1) {  
 JOptionPane.showMessageDialog(null, "修改成功");  
   
 this.reSetValue();//清楚留在表格中的数据  
   
 this.fillTable(new BeanCptInfo());//在上面的table中显示已经更新过的信息  
 }else {  
 JOptionPane.showMessageDialog(null, "修改失败");  
 }  
 } catch (Exception e) {  
 // TODO 自动生成的 catch 块  
 e.printStackTrace();  
 }   
 }

## 管理员：

### **查询竞赛报名情况**：

**其中要显示的属性有:** 赛项名称, 竞赛名称, 学生姓名, 学号, 队伍编号, 专业, 年级, 班级, 报名时间, 报名状态

**用到的表有:** cpt\_info, cpt\_itm\_info, stu\_info, team, team\_up, register

select stu\_info.StuId, stu\_info.Name, team.TeamId, cpt\_info.CptName, cpt\_itm\_info.CptItemName, stu\_info.SClass, stu\_info.Grade, stu\_info.Profess, register.RegiTime, register.RegiStatus
  
from cpt\_info,cpt\_itm\_info,stu\_info,team,team\_up,register
  
where team\_up.TeamId = team.TeamId and team\_up.StuId = stu\_info.StuId and team.TeamId = register.TeamId and register.CptId = cpt\_info.CptId and cpt\_info.CptItemId = cpt\_itm\_info.CptItemId and cpt\_info.CptName like '% %' and cpt\_info.HoldYear = ' ';

### 查询竞赛获奖信息：

**其中要显示的属性有:** 赛项名称, 竞赛名称, 学号, 学生姓名,班级, 年级, 专业, 队伍编号, 获奖等级

**用到的表有:** cpt\_info, cpt\_itm\_info, stu\_info, team, team\_up

select stu\_info.StuId, stu\_info.Name, team.TeamId, cpt\_info.CptName, cpt\_itm\_info.CptItemName, stu\_info.SClass, stu\_info.Grade, stu\_info.Profess, team.RwdGrade
  
from cpt\_info,cpt\_itm\_info,stu\_info,team,team\_up
  
where team\_up.TeamId = team.TeamId and team\_up.StuId = stu\_info.StuId and cpt\_info.CptItemId = cpt\_itm\_info.CptItemId and team.IsFinCpt = '是';

### 查询学生获奖情况：

**其中要显示的属性有:** 学号, 学生姓名, 竞赛名称, 获奖等级,专业, 年级, 班级

**要用到的表有:** stu\_info,team\_up, team, register, cpt\_info

StringBuffer sb = new StringBuffer("select stu\_info.StuId, stu\_info.Name, cpt\_info.CptName, team.RwdGrade, stu\_info.Profess, stu\_info.Grade, stu\_info.SClass from stu\_info, team, team\_up, cpt\_info, register where stu\_info.StuId=team\_up.StuId and team.TeamId=team\_up.TeamId and team\_up.TeamId = register.TeamId and cpt\_info.CptId = register.CptId and team.IsFinCpt = '是' and stu\_info.StuId=" + stuInfo.getStuId() " and stu\_info.Name like '%" + stuInfo.getName() + "%' and stu\_info.Profess='"+ stuInfo.getProfess() +"' and stu\_info.Grade='" + stuInfo.getGrade() + "'" );

一定要注意如果输入的是中文, 那么一定要加上单引号'', 不然在查询的时候会出错

定义StringBuffer的好处就是可以将sql语句分开定义, 可以使用StringBuffer的函数进行对sql语句的操作

比如: 将where之后的判断条件分开, 从而分多种情况讨论

StringBuffer sb = new StringBuffer("select \* from stu\_info");
  
//现在就是将sql语句视作为一串字符串, 之后sb对象对这串字符串可以进行添加等操作
  
   
BeanStuInfo stu = new BeanStuInfo();
  
   
if(stu.StuId is not null){
  
 sb.append(" where StuId = '32001014'");//要注意where前的空格,是sql语句的空格
  
}
  
else if(stu.StuId is null){
  
 sb.append(" where StuId = '32001020'");
  
}
  
   
PreparedStatement pstmt = conn.prepareStatement(sb.toString());//这里注意要将StringBuffer类型进行转化,转化为String类型

在查询时, 有多个查询输入框, 此时并不一定要输入全部信息才能进行查询, 单独输入一个信息也可以进行关于该信息的查询

对于sql语句的处理方式, 采用一种替换replaceFirst的方式, 为了避免条件多了之后判断复杂

现在是两个输入框, 所以有四种情况, 分别对其做出判断, 以保证每一种情况都能有对应的查询信息

以查询竞赛报名信息为例 :



### 统计某段时间内获奖数量最多的学生:

**一个学生可以参加多个不同竞赛的同一队伍, 但是不能参加同一个竞赛的多个队伍**

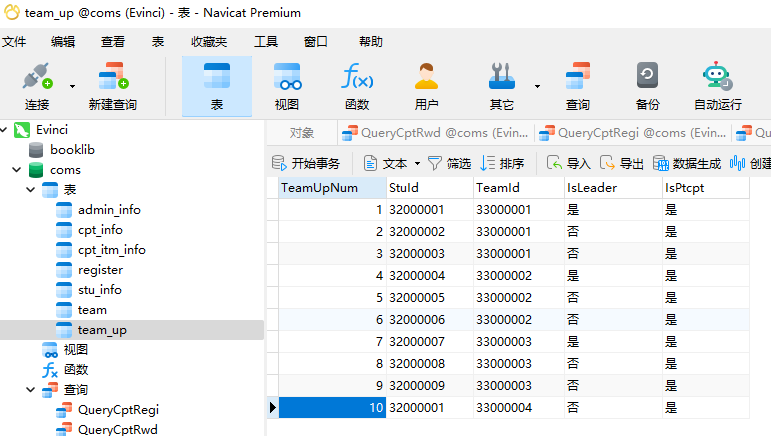
aaa学生参加了微积分竞赛的吃的很撑小组

aaa学生参加了流体力学的吃的很撑小组

**(Wrong)解决思路一:**

team\_up表中需要学号信息可以重复，因为**学生可以报不止一个队伍**

所以进行重构, 加上了一个组队编号, 使得学号不再是主键, 所以可以重复



**(Right)解决思路二:**

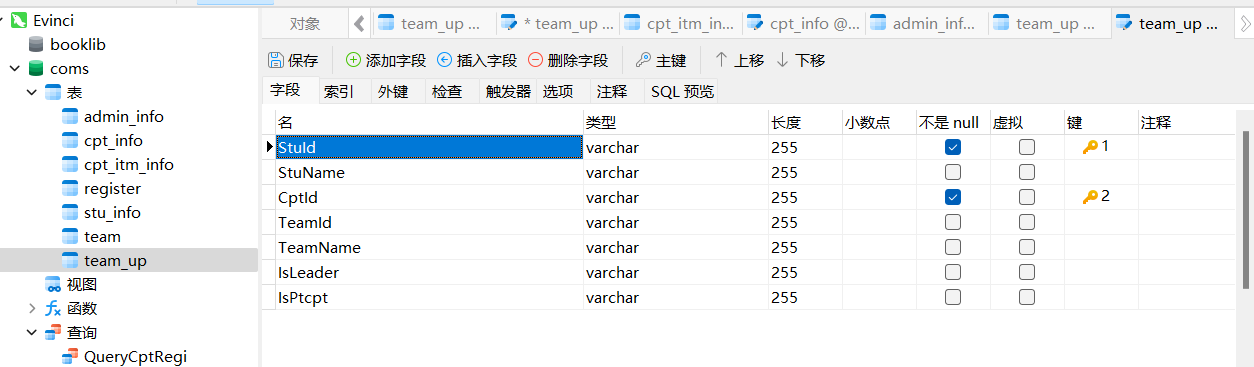
**一个学生只能参加一个竞赛中的一个队伍, 但是一个队伍可以参加不同的竞赛**

即一个学生可以在相同的队伍中既参加数学竞赛也参加物理竞赛

所以学生编号和竞赛编号可以重复但不能同时重复

所以将学号和竞赛编号设置为主键

不用再添加TeamUpNum使其作为主键, 这样三个编号属性都不是主键在判断上会出现问题



将学号和竞赛编号设置为主键



以如上字段构建team\_up表

统计获奖个数也就是多条记录的获奖级别

**其中要显示的属性有**:获奖个数, 学号, 学生姓名, 专业, 年级, 班级

**涉及到的表有:** stu\_info, team\_up, team,cpt\_info, register

select count(team.RwdGrade) RwdNum,stu\_info.StuId, stu\_info.Name, stu\_info.Profess, stu\_info.Grade, stu\_info.SClass
  
from stu\_info, team, team\_up, cpt\_info, register
  
where stu\_info.StuId=team\_up.StuId and team.TeamId=team\_up.TeamId and team\_up.TeamId = register.TeamId and cpt\_info.CptId = register.CptId and team.IsFinCpt = '是' and stu\_info.Grade = '大三'
  
group by stu\_info.StuId
  
order by RwdNum desc

查询特定时间: 因为竞赛之间的举行时间是间隔且不定期且没有时间规律的, 所以以竞赛为目标, 去查找特定时间段在所有竞赛获取的奖项相对于学生来说现实意义不大

这时**以学生的学龄为时间段查询**, 是比较有现实意义且使用频率较多的一种方法

比如大三学年, 获奖学生最多的学生

但是如果要查询指定某年几月到某年几月的获奖信息, 这样的情况在现实中也是会出现的, 并且出现频率较高, 所以还是要进行时间段的查询

### 统计特定竞赛获奖学生的信息

含有属性应有：竞赛名称, 获奖等级, 学号, 学生姓名,队伍编号, 队伍名称, 专业, 年级, 班级,

所用到的表有: cpt\_info, stu\_info, team, register, team\_up

即以竞赛名称进行模糊搜索, 获取该场竞赛中所有获奖的学生信息, 并**以获奖等级降序排序**

可以根据竞赛和专业进行查询

select cpt\_info.CptName,team.RwdGrade, stu\_info.StuId, stu\_info.Name, stu\_info.Profess, stu\_info.Grade, stu\_info.SClass, team.TeamId, team.TeamName
  
from cpt\_info,stu\_info,team,register,team\_up
  
where stu\_info.StuId = team\_up.StuId and team\_up.TeamId = team.TeamId and team.TeamId = register.TeamId and register.CptId = cpt\_info.CptId and cpt\_info.CptName like '%微%' and stu\_Info.Grade = '大三'

### 统计特定时间某个专业的获奖学生的信息

**这里的时间同样认为是以学龄为单位**, 例如大三或大二

即以学龄和专业为搜索, 获取该条件中所有竞赛的学生的获奖信息,并**按照获奖数量和获奖等级排序**

**其中要显示的属性有**:获奖数量, 获奖等级, 学生姓名, 竞赛名称,竞赛编号, 学号, 队伍编号, 队伍名称, 专业, 年级, 班级

**涉及到的表有:** stu\_info, team\_up, team,cpt\_info, register

可以根据队伍编号和专业进行查询

select count(team.RwdGrade) RwdNum,team.RwdGrade, stu\_info.Name, cpt\_info.CptName, cpt\_info.CptId, stu\_info.StuId, team.TeamId, team.TeamName, stu\_info.Profess, stu\_info.Grade, stu\_info.SClass
  
from stu\_info, team, team\_up, cpt\_info, register
  
where stu\_info.StuId=team\_up.StuId and team.TeamId=team\_up.TeamId and team\_up.TeamId = register.TeamId and cpt\_info.CptId = register.CptId and team.IsFinCpt = '是' group by stu\_info.StuId order by RwdNum desc, team.RwdGrade

## 学生：

### 学生竞赛组队：

两个显示，一个提交组队信息,也即是队员信息

一个显示显示的是队伍信息team表，显示还没有完赛的队伍信息

select \* from team where IsFinCpt = '否';

一个显示的是自己的报名信息, 同一个竞赛一个人只能报名一个队伍, 在填写组队(报名)信息之后, 数据库中的报名信息变化

组队报名的流程 :

首先由队长报名, 在team\_up中创建队伍同时也是提交自己的队员信息, 然后将队伍信息公布, 学生填写组队信息

组队信息中的属性有以下几个, 即一个队员表, 组队表就相当于一个队员表

学号, 队伍编号, 竞赛编号

学生功能两个迷惑点: 一个是队长怎么创建队伍, 然后报名

创建队伍的属性: 队伍编号, 队伍名称, 队伍人数, 赛项编号, 竞赛编号

怎么获取登陆学生输入的学号信息, 从而一个主面板对应一个学生

之后在该学生创建队伍的时候, 同步把自己的信息添加进去(队员信息):学号, 名字, 是否是队长, 在是否参赛这显示一定是参赛的, 同时把 赛项编号和竞赛编号也一并录入进去

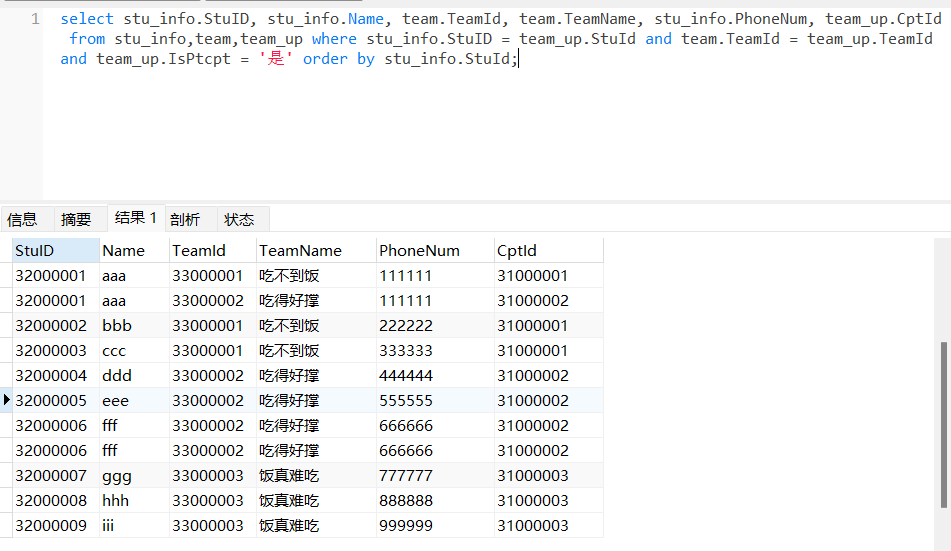
学生组队一张里面显示表, 一张添加表, 显示和添加的都是组队信息

**显示信息:**

其中拥有属性: 学号, 姓名, 队伍编号,队伍名称, 竞赛编号

要用到的表有:😛stu\_info, team, team\_up

select stu\_info.StuId, stu\_info.Name, team.TeamId, team.TeamName, stu\_info.PhoneNum, team\_up.CptId from stu\_info,team,team\_up where stu\_info.StuID = team\_up.StuId and team.TeamId = team\_up.TeamId and team\_up.IsPtcpt = '是' order by stu\_info.StuId;





**添加信息:**

因为一个学生只能在一个竞赛中只能参加一个队伍, 所以在组队的时候必须要加上竞赛信息, 即竞赛表编号, 来唯一确定一个竞赛, 从而唯一确定一个学生的在一个竞赛中的组队信息

可能出现一个学生参加不同的队伍, 但是参加同一个竞赛

这个判断筛查是在**添加组队信息**的时候需要进行的

在进行添加判断的时候, 提取添加的信息输入, 进行**StuId和CptId分别是否相等的判断,** 因为StuId和CptId**唯一确定一个学生参加竞赛的情况**

### 学生竞赛报名：

### 查询竞赛信息：

### 查询获奖信息：

张三和李四互相参加对方的队伍,参加同一个竞赛, 但是报名参赛的时候不能同时成功

又创建了一个player表, 设置学号和竞赛编号为主键, 希望可以用来进行不同同时成功的条件判断, 但这样player表就要取代team\_up表的位置

查询某一天的竞赛获奖情况, 某一天就是竞赛的开始时间

设置奖项需要进行时间比较, 要得到当前报名的时间, 报名表

SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy-MM-dd");
  
String time = sdf1.format(System.currentTimeMillis());

那team\_up表用来干什么呢, 用来作为组队信息, 便于管理员查询组队信息

## 应该在一开始就在navicat里面把所有的操作都模拟一遍的, 因为这个系统本质上就是对数据库的操作

* **管理对象之间的关系就是表的定义和表之间的关系,**
* **管理对象的信息就是表中的属性信息 ,**
* **对信息的增删改查就是sql语句**

在数据库中全部演示过, 都觉得可以了之后, 然后根据MVC构建模式构建model view control

model就是用Java对于数据库中表的呈现

再写control, 也就是要做的一系列对数据的操作, 也就是sql语句

最后再写view, 根据功能学习一下怎么把数据显示到页面山, 把页面设计一下就好

而这次写的过程恰好把关键的第二三步反过来, 先弄的界面, 界面弄好了才去弄得功能

原因是用了windowsBuilder这个插件, 对于这个东西完全不熟悉, 以为这个的操作才是重点, 在界面设计上花费了比较多的时间

事实是功能没有想好, 界面弄得也很慢

**一切都是要把功能放在第一位**,然后再去考虑怎么在界面上显示的问题

当然也要注意 显示的形式 和 功能展示结果的形式 的一致性

1. 两个学生参加同一个竞赛的不同队伍, 都进行报名, 应该出现"在一项竞赛中, 学生只能作为一个队伍的队员"---报名的条件判断检验
2. 根据竞赛的报名时间和竞赛时间信息, 管理员设置获奖等级的时间应该晚于竞赛时间, 学生报名的时间应该早于报名截止时间晚于报名开始时间---报名和获奖的时间判断检验
3. 根据时间段进行查询竞赛获奖学生信息

对于时间的操作:

**在java中,进行两种格式的转换:**

* 插入数据库时, **存入当前的日期**, 需要进行格式的转换

import java.text.SimpleDateFormat;
  
   
SimpleDateFormat formatter = new SimpleDateFormat( "yyyy-MM-dd ");
  
String date = formatter.format(new Date());//格式化数据，取当前时间

* String格式的数据转化为Date格式

import java.text.SimpleDateFormat;
  
   
SimpleDateFormat formatter = new SimpleDateFormat( "yyyy-MM-dd ");
  
String s = "2011-07-09 ";
  
Date date = formatter.parse(s);

**在sql中进行格式的转换:**

cast(要转换的变量 as 要转换的格式)

cast(cpt\_info.time as date) //cpt\_info的类型是String

# 重大改变!

现在又进行了表的更改, 把队伍表中的获奖信息移除, 因为一个队伍可以参加多个竞赛, 同时为了确定获奖信息授予的是否完赛属性也可以移除了

把获奖信息添加到报名表, 因为这样才能唯一确定一个竞赛的报名情况, 虽然这样听起来不是很合理, 报名表里面有一个获奖信息, 但是功能上可以设计

如果更合理一点的就再创建一个获奖队伍信息表, 从而每一项获奖记录都有与之对应的竞赛和队伍乃至个人

那现在要修改的就是和register,team相关联的所有代码和sql语句中的变量全部改掉

跟获奖功能有关的sql语句全部改掉

先从Bean开始改, 然后Dao, 最后View

## 现在要把所有关于获奖的功能的sql语句改变

查询学生获奖信息

select stu.StuId,stu.Name,team\_up.TeamId,cpt.CptName,cptitem.CptItemName,stu.SClass,stu.Grade,stu.Profess,register.RwdGrade
  
from stu\_info stu,cpt\_info cpt,cpt\_item\_info cptitem,team\_up teamup,register regi
  
where team\_up.StuId = stu\_info.StuId and team\_up.StuId = register.StuId and register.CptId = cpt\_info.CptId and cpt\_info.CptItemId = cpt\_itm\_info.CptItemId

已经全部写在navicat中了

一个数据库设计中的重要问题：

表与表之间主键的设置和外键的关联，与管理员权限的矛盾

有了外键，属性不能被任意删除，那管理员管理删除就没有完全的权限

管理员就是应该合理的进行删除, 所以外键和主键的合理设置一定是基础

## 其实现在我又把竞赛奖项设置和是否完赛这两个属性又拿到队伍中去了,尽管知道会有些小问题, 但是在功能界面已经做好的前提下再变更最基础的表属性,之后要做的修改真的要把人磨死,就等于从头做一遍

那现在要做的就是写一个设置判断报名时间的