

**报告名称： 数据库原理C实验报告**

**班 级： 231202**

**学 号： 20201000128**

**姓 名： 刘瑾瑾**

目录

[实验一 1](#_Toc23248)

[一、 实验目的与要求 1](#_Toc28479)

[1.1实验目的 1](#_Toc542)

[1.2 实验要求 1](#_Toc942)

[1.3 实验数据库中的表 1](#_Toc4194)

[二、实验代码 2](#_Toc13903)

[2.1 单表查询 2](#_Toc8155)

[2.2 连接查询（自身连接） 2](#_Toc21225)

[2.3连接查询（多表连接） 2](#_Toc9625)

[2.4 分组统计查询 2](#_Toc31021)

[2.5 课本内容： 3](#_Toc11772)

[三、运行结果 3](#_Toc2462)

[3.1 单表查询 3](#_Toc11893)

[3.2 连接查询（自身连接） 3](#_Toc26344)

[3.3 连接查询（多表连接） 3](#_Toc7708)

[3.4 分组统计查询 3](#_Toc5770)

[3.5 课本内容 4](#_Toc21519)

[四、问题解决 4](#_Toc25670)

[4.1 问题一  4](#_Toc25019)

[4.2 问题二  5](#_Toc1471)

[4.3 问题三  5](#_Toc4221)

[五、实验总结 6](#_Toc26646)

[实验二 7](#_Toc23248)

[一、实验目的与要求 7](#_Toc9534)

[1.1实验目的 7](#_Toc24475)

[1.2 实验要求 7](#_Toc25078)

[二、实验代码 7](#_Toc29122)

[2.1 带有IN谓词的子查询 7](#_Toc15415)

[2.2带有比较运算符的子查询 7](#_Toc29690)

[2.3 带有ANY(SOME)或ALL谓词的子查询 8](#_Toc14646)

[2.4带有EXISTS谓词的子查询 8](#_Toc20004)

[2.5 并操作UNION 9](#_Toc8935)

[2.6交操作INTERSECT 9](#_Toc26027)

[2.7差操作EXCEPT 9](#_Toc10020)

[三、运行结果 10](#_Toc7834)

[3.1 带有IN谓词的子查询 10](#_Toc14594)

[3.2带有比较运算符的子查询 10](#_Toc2593)

[3.3 带有ANY(SOME)或ALL谓词的子查询 10](#_Toc3507)

[3.4带有EXISTS谓词的子查询 10](#_Toc22109)

[3.5 并操作UNION 10](#_Toc6577)

[3.6交操作INTERSECT 10](#_Toc29035)

[3.7差操作EXCEPT 11](#_Toc22534)

[四、问题解决 11](#_Toc8718)

[4.1 问题一  11](#_Toc13533)

[4.2 问题二  12](#_Toc10855)

[五、实验总结 12](#_Toc32215)

[实验三 13](#_Toc28050)

[一、实验目的与要求 13](#_Toc3499)

[1.1实验目的 13](#_Toc5938)

[1.2 实验要求 13](#_Toc22041)

[二、实验代码 13](#_Toc31776)

[2.1 插入元组 13](#_Toc16963)

[2.2 插入子查询结果 13](#_Toc26082)

[2.3 修改某一个元组的值 14](#_Toc31964)

[2.4 修改多个元组的值 14](#_Toc18072)

[2.5 带子查询的修改语句 14](#_Toc16598)

[2.6删除某一个元组的值 14](#_Toc376)

[2.7 删除多个元组的值 14](#_Toc20594)

[2.8 带子查询的删除语句 14](#_Toc5143)

[三、运行结果 15](#_Toc19935)

[3.1 插入元组 15](#_Toc3856)

[3.2 插入子查询结果 15](#_Toc18896)

[3.3 修改某一个元组的值 15](#_Toc200)

[3.4 修改多个元组的值 16](#_Toc16297)

[3.5 带子查询的修改语句 16](#_Toc19291)

[3.6删除某一个元组的值 16](#_Toc13366)

[3.7 删除多个元组的值 16](#_Toc26365)

[3.8 带子查询的删除语句 16](#_Toc4881)

[四、问题解决 17](#_Toc23631)

[4.1 问题一  17](#_Toc2067)

[五、实验总结 17](#_Toc24190)

[实验四 18](#_Toc15757)

[一、实验目的与要求 18](#_Toc27421)

[1.1实验目的 18](#_Toc27939)

[1.2 实验要求 18](#_Toc17952)

[二、实验代码 18](#_Toc24685)

[2.1 创建行列子集视图 18](#_Toc6756)

[2.2创建带虚拟列的视图 18](#_Toc10202)

[2.3创建分组视图 19](#_Toc2970)

[2.4创建带WITH CHECK OPTION的行列子集视图 19](#_Toc16737)

[2.5 查询视图 19](#_Toc11428)

[2.6删除视图 19](#_Toc30213)

[2.7删除视图 19](#_Toc9621)

[三、运行结果 20](#_Toc11324)

[3.1 创建行列子集视图 20](#_Toc28405)

[3.2创建带虚拟列的视图 20](#_Toc7855)

[3.3创建分组视图 20](#_Toc29698)

[3.4创建带WITH CHECK OPTION的行列子集视图 20](#_Toc8695)

[3.5 查询视图 21](#_Toc15353)

[3.7删除视图 21](#_Toc14657)

[四、问题解决 21](#_Toc31408)

[4.1 问题一  21](#_Toc30969)

[4.2 问题二  21](#_Toc11023)

[五、实验总结 21](#_Toc28654)

[实验五 23](#_Toc10782)

[一、实验目的与要求 23](#_Toc21141)

[1.1实验目的 23](#_Toc16329)

[1.2实验要求 23](#_Toc10894)

[二、实验步骤 23](#_Toc2517)

[三、需求分析 24](#_Toc1353)

[3.1表 24](#_Toc23898)

[3.2 整体E-R图 24](#_Toc23898)

[3.3关系模型 25](#_Toc32630)

[3.4权限设计 25](#_Toc999)

[3.5功能需求 25](#_Toc14647)

[四、功能设计 26](#_Toc2187)

[4.1登录部分 26](#_Toc26419)

[4.2 学生端部分 27](#_Toc22014)

[4.2 教师端部分 27](#_Toc22014)

[4.4密码修改部分 35](#_Toc22341)

[五、效果呈现 36](#_Toc24744)

[六、问题解决 41](#_Toc13256)

[6.1问题一 41](#_Toc15643)

[6.2问题二 42](#_Toc25804)

[七、实验总结 42](#_Toc1631)

**实验一**

1. **实验目的与要求**

**1.1实验目的**

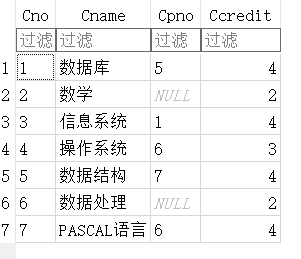
1. 掌握SQL程序设计基本规范，熟练运用SQL语言实现数据基本查询，包括单表查询、分组统计查询、连接查询
2. 能按照SQL程序设计规范写出具体的SQL查询语句，并调试通过

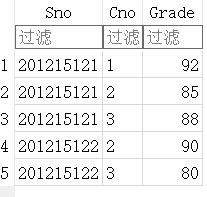
**1.2 实验要求**

1. 设计各种单表查询SQL语句；
2. 设计分组统计查询语句；
3. 设计单表针对自身的连接查询；
4. 设计多表的连接查询。

**1.3 实验数据库中的表**

1. Student表
2. Course表



1. SC表

**二、实验代码**

**2.1 单表查询**

查询“刘晨”的学号与姓名：

select Sno,Sname

from Student

where Sname='刘晨';

**2.2 连接查询（自身连接）**

查询每门课的间接先修课：

select first.Cno,second.Cpno

from Course first,Course second

where first.Cpno=second.Cno

**2.3连接查询（多表连接）**

查询每个学生的学号、姓名、选修的课程名及成绩：

select Student.Sno,Sname,Cname,Grade

from Student,SC,Course

where Student.Sno=SC.Sno and SC.Cno=Course.Cno;

**2.4 分组统计查询**

查询平均成绩大于等于80的学生学号：

select Sno,avg(Grade)

from SC

group by Sno

having avg(Grade)>=80;

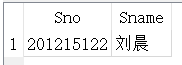
**2.5 课本内容：**

查询全体学生的姓名、出生年份和所在院系，要求用小写字母表示系名：

select Sname,"Year of Birth",2014-Sage,LOWER(Sdept)

from Student;

**三、运行结果**

**3.1 单表查询**

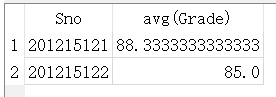
**3.2 连接查询（自身连接）**

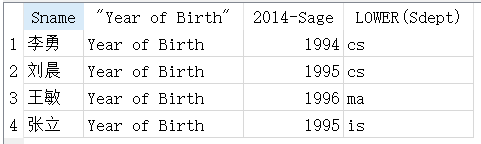


**3.3 连接查询（多表连接）**



**3.4 分组统计查询**



**3.5 课本内容**

**四、问题解决**

**4.1 问题一**

进行连接查询（自身连接）时，代码如下：

select first.Cno,second.Cpno

from Course first,Course second

where first.Cpno=sercond.Cno

执行结果如下：

执行已完成，但有错误。

结果: no such column: sercond.Cno

**4.1.1 解决方法**

经检查，是由于给定Course表的别名为seccond,而进行连接时多打

了一个r,错写为sercond,导致进行连接时出错。

**4.1.2 运行结果**

**4.1.3 调试分析总结**

遇到问题时，要根据提示信息认真检查代码，比如上述的执行结果中： no such column: sercond.Cno，提示没有sercond.Cno属性列，连接的条件 出现问题。 大部分SQL语句执行错误可能只是某处的大小写写错或 者多 写了一个字母，导致前后表的名称不一致。

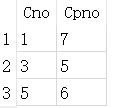
在多表连接中如果不存在上述问题，我们应检查是否存在同名的属性 列，可能是没有加表名前缀，导致出现识别错误。

**4.2 问题二**

在问题一的基础上，我发现间接先行课为空的课程号也会显示出，思考如 何只显示有间接先行课的课程号。

**4.2.1 解决方法**

在条件中加入second.Cpno is not null即可解决。

**4.2.2 运行结果**

**4.2.3 调试分析总结**

当查询结果不符号设计要求时，可以思考SQL语句的逻辑，增加筛 选的 条件或者考虑其他方式重新设计SQL语句。

**4.3 问题三**

进行分组查询时，代码如下：

select Sno,avg(Grade）

from SC

group by Sno

having avg(Grade)>=80;

执行结果如下：

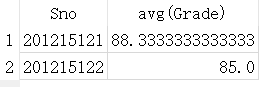
执行已完成，但有错误。

结果: near "from": syntax error

**4.3.1 解决方法**

经检查，源代码第一行中Grade前后括号不一致，左括号为英文括号， 右括号为中文括号，将其修改后即可正常执行。

**4.3.2 运行结果**

****

**4.3.3 调试分析总结**

当SQL语句执行出现错误，首先要根据提示信息进行检查，大部分情 况都可以根据提示找到问题所在，如果检查不到错误，可能是查询逻辑存 在问题，再去思考所编写的语句是否符合SQL设计规范。

**五、实验总结**

在本次实验中，我分别使用sqlite3和DB Browser(SQLite)创建了Student表、Course表和SC表：sqlite3创建表格需在命令行使用CREATE动词，定义新的表格，当出现语法错误或拼写错误，表格创建容易失败，再次创建又需要重新编写SQL语句，表格元组需单个插入，繁琐且效率低，可视化效果差，但是可以让我们对SQL语句的使用更加熟练；而在DB Browser（SQLite）只需创建表，给定表名，勾选相关属性即可创建成功，可直接浏览数据，并在表格中直接添加数据，方便快捷。

通过单表查询、多表连接查询和分组查询SQL语句的实际编写和运行，我对SQL的查询操作更加熟悉。进行查询时，书写SQL语句要注意大小写的一致性以及括号的匹配性这些基本程序问题；多表连接时，重名的属性列要加上表名前缀；

查询结果不符合设计要求时，要思考如何修改查询条件。

**实验二**

**一、实验目的与要求**

**1.1实验目的**

1. 掌握SQL程序设计基本规范，熟练运用SQL语言实现数据高级查询，包括各种嵌套语句和聚合查询语句；
2. 熟悉并掌握IN、ALL、ANY、EXISTS等谓词的使用，熟悉并操作UNION、交操作INTERSECT和差操作EXCEPT的使用；
3. 分析查询需求，能按照SQL程序设计规范写出具体的SQL查询语句，并调试通过。

**1.2 实验要求**

(1)设计各种嵌套查询；

(2)设计集合查询语句。

**二、实验代码**

**2.1 带有IN谓词的子查询**

查询与“刘晨”在同一个系学习的学生：

select Sno,Sname,Sdept

from Student

where Sdept in

(select Sdept

from Student

where Sname='刘晨');

**2.2带有比较运算符的子查询**

找出每个学生超过他自己选修课程平均成绩的课程号：

select Sno,Cno

from SC x

where Grade>=(

select avg(Grade)

from SC y

where y.Sno=x.Sno);

**2.3 带有ANY(SOME)或ALL谓词的子查询**

查询非计算机科学系中比计算机科学系任意一个学生年龄小的学生姓名和年龄：

select Sname,Sage

from Student

where Sage<ANY(

select Sage

from Student

where Sdept='CS'

)

and Sdept<>'CS';

等效于：

select Sname,Sage

from Student

where Sage<(

select MAX(Sage)

from Student

where Sdept='CS'

)

and Sdept<>'CS';

**2.4带有EXISTS谓词的子查询**

查询所有选修了1号课程的学生姓名：

select Sname

from Student

where EXISTS

(SELECT \*

from SC

where Sno=Student.Sno and Cno='1');

**2.5 并操作UNION**

查询计算机科学系的学生及年龄不大于19岁的学生：

select \*

from Student

where Sdept='CS'

union

select \*

from Student

where Sage<=19;

**2.6交操作INTERSECT**

查询计算机科学系的学生与年龄不大于19岁的学生的交集：

select \*

from Student

where Sdept='CS'

intersect

select \*

from Student

where Sage<=19;

**2.7差操作EXCEPT**

查询计算机科学系的学生与年龄不大于19岁的学生的差集：

select \*

from Student

where Sdept='CS'

except

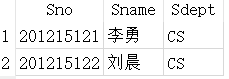
select \*

from Student

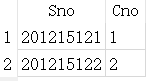
where Sage<=19;

**三、运行结果**

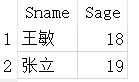
**3.1 带有IN谓词的子查询**



**3.2带有比较运算符的子查询**

****

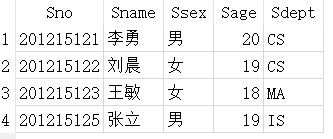
**3.3 带有ANY(SOME)或ALL谓词的子查询**

****

**3.4带有EXISTS谓词的子查询**



**3.5 并操作UNION**



**3.6交操作INTERSECT**



**3.7差操作EXCEPT**

**四、问题解决**

**4.1 问题一**

源代码：

select Sname,Sage

from Student

where Sage<ANY(

select Sage

from Student

where Sdept='CS'

)

and Sdept<>'CS';

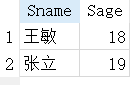
执行结果：

执行已完成，但有错误。

结果: near "select": syntax error

**4.1.1 解决方法**

经检查，源代码无误，请教老师，发现该代码在DB Browser(SQLite)无 法执行，而在MySQL中可以正常执行。DB Browser(SQLite)不支持带有ANY （SOME）或ALL谓词的子查询。重新设计SQL语句，使用集函数替代，可 以正常执行。

**4.1.2 运行结果**

**4.1.3 调试分析总结**

当SQL语句语法检查无误且符合设计规范时，可以尝试思考是否是运行 环境的问题，当在其他运行环境中可以运行，说明设计的SQL语句依然具有 正确性，只是运行环境不满足要求。

**4.2 问题二**

验证课本基于派生表的查询源代码：

查询每个学生超过他自己选修课程平均成绩的课程号：

select Sno,Cno

from SC,(select Sno,avg(Grade) from SC group by Sno)

AS Avg\_sc(avg\_sno,avg\_grade)

where SC.Sno=Avg\_sc.avg\_sno and SC.Grade>Avg\_sc.avg\_grade;

执行结果：

执行已完成，但有错误。

结果: near "(": syntax error

**4.2.1 解决方法**

经验证，基于派生表的查询操作可以在MySQL上正常执行，故同问题一， 是运行环境的问题。

**4.2.2 调试分析总结**

当设计的SQL语句不是常规的操作时，如涉及到一些关键词或者派生表， 如果执行错误，可以尝试使用其他数据库操作软件进行执行，可能只是运行环 境不支持。

**五、实验总结**

在本实验中，主要对嵌套查询和集合查询进行了操作。嵌套查询可以使用户使用多个简单查询构成复杂查询，增强SQL的查询能力，使其具有更好的结构化。集合查询可以对多个SQL查询语句的结果进行集合操作，但有严格的要求，每个SQL的列数必须相同，对应的数据类型也必须相同。

当SQL嵌套语句语法无误且符合设计规范仍然无法运行时，我们应考虑使用的数据库操作软件是否支持对应谓词的使用，使用其他的数据库操作软件如MySQL进行验证。

**实验三**

**一、实验目的与要求**

**1.1实验目的**

1. 熟悉数据库的数据更新操作，能够熟练运用SQL语言实现数据更新操作，包括插入、修改和删除；
2. 理解和掌握insert、update和delete关键词的使用，熟悉其应用；
3. 能按照SQL程序设计规范写出具体的SQL操作语句，并调试通过。

**1.2 实验要求**

(1)设计单元组插入、批量数据插入等SQL操作语句；

(2)设计修改数据的SQL操作语句；

(3)设计删除数据的SQL操作语句。

**二、实验代码**

**2.1 插入元组**

将一个新学生元组（学号：201215128，姓名：陈冬，性别：男，所在系：IS，年龄：18岁）插入到Student表中：

insert

into Student(Sno,Sname,Ssex,Sdept,Sage)

values('201215128','陈冬','男','IS',18);

**2.2 插入子查询结果**

对每一个系，求学生平均成绩，并把结果存入数据库：

create table Dept\_age

(Sdept CHAR(15) primary key,

Avg\_age SMALLINT);

insert

into Dept\_age(Sdept,Avg\_age)

select Sdept,avg(Sage)

from Student

group by Sdept;

select \*

from Dept\_age

**2.3 修改某一个元组的值**

将学生201215121的年龄改为22岁：

update Student

set Sage=22

where Sno='201215121';

**2.4 修改多个元组的值**

将所有学生的年龄增加一岁：

update Student

set Sage=Sage+1;

**2.5 带子查询的修改语句**

将计算机科学系全体学生的成绩置零：

update SC

set Grade=90

where Sno in(

select Sno

from Student

where Sdept='CS');

**2.6删除某一个元组的值**

删除学号为201215128的学生记录：

delete

from Student

where Sno='201215128';

**2.7 删除多个元组的值**

删除所有学生的选课记录：

delete

from SC;

**2.8 带子查询的删除语句**

删除计算机科学系所有的学生选课记录：

delete

from SC

where Sno in(

select Sno

from Student

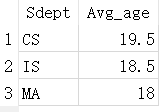
where Sdept='CS');

**三、运行结果**

**3.1 插入元组**

|  |  |
| --- | --- |
|  |  |

**3.2 插入子查询结果**

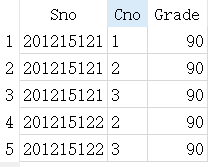


**3.3 修改某一个元组的值**

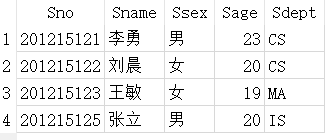
|  |  |
| --- | --- |
|  |  |

**3.4 修改多个元组的值**

**3.5 带子查询的修改语句**



**3.6删除某一个元组的值**



**3.7 删除多个元组的值**

执行完成。

结果: 0 行返回，耗时 8ms

注：此时SC表无元组，为空白表

**3.8 带子查询的删除语句**

|  |  |
| --- | --- |
|  |  |

**四、问题解决**

**4.1 问题一**

执行SQL插入语句无法执行：

源代码：

insert

into Student(Sno,Sname,Ssex,Sdept,Sage)

values('201215128','陈冬','男','IS',18);

执行结果;

执行已完成，但有错误。

结果: UNIQUE constraint failed: Student.Sname

**4.1.1 解决方法**

经检查，之前执行过插入操作，该元组已经在Student存在，故该元组不 可重复插入，可将其删除，重新插入。

**4.1.2 运行结果**

**4.1.3 调试分析总结**

当执行增、删、改操作的SQL语句无误但无法执行时，我们应该考虑是否违背了完整性约束条件：实体完整性、参照完整性和用户定义完整性，例如删除学号违背了实体完整性，添加重复的名字违背了用户定义完整性。

**五、实验总结**

在本次实验中，我设计了增加、删除和修改数据的SQL语句，执行了对应的操作。与查询操作不同，增加、删除和修改操作会改变数据库中的数据，所以需要考虑完整性约束条件。另外，注意插入语句：当INTO子句中只指出表名，没有指出属性名，新元组要在表的所有属性列上都指定值，属性列的次序与CREATE TABLE中的次序相同，VLAUES子句对新元组的各属性列赋值，一定要注意值与属性列要一一对应。

**实验四**

**一、实验目的与要求**

**1.1实验目的**

1. 掌握SQL程序设计基本规范，熟练运用SQL语言实现视图的建立、删除、查询和更新操作；
2. 明确行列子集视图、带虚拟列的视图和分组视图的概念
3. 区分带with check option的视图与普通视图；
4. 分析设计需求，能按照SQL程序设计规范写出具体的SQL视图操作语句，并调试通过。

**1.2 实验要求**

(1)分别创建行列子集视图、带虚拟列的视图和分组视图；

(2)创建视图和带with check option的视图，验证视图with check option选项的有 效性；

(3)对创建好的视图进行查询、更新和删除操作。

**二、实验代码**

**2.1 创建行列子集视图**

建立计算机科学系学生的视图：

create view IS\_Student

as

select Sno,Sname,Sage

from Student

where Sdept='CS';

**2.2创建带虚拟列的视图**

定义一个反映学生出生年份的视图：

create view BT\_S(Sno,Sname,Sbirth)

as

select Sno,Sname,2014-Sage

from Student;

**2.3创建分组视图**

将学生的学号及平均成绩定义为一个视图：

create view S\_G(Sno,Gavg)

as

select Sno,avg(Grade)

from SC

group by Sno;

**2.4创建带WITH CHECK OPTION的行列子集视图**

建立计算机科学系学生的视图，并要求进行修改和插入操作时仍需保证该视图只有计算机科学系的学生：

create view IS\_Student

as

select Sno,Sname,Sage

from Student

where Sdept='CS';

with check option

**2.5 查询视图**

在计算机科学系学生的视图中找出年龄小于20岁的学生

select Sno,Sage

from IS\_Student

where Sage<20;

**2.6删除视图**

update IS\_Student

set Sname="刘辰"

where Sno="201215122";

**2.7删除视图**

删除计算机科学系学生的视图：

drop view IS\_Student

**三、运行结果**

**3.1 创建行列子集视图**

|  |  |
| --- | --- |
|  |  |

**3.2创建带虚拟列的视图**

|  |  |
| --- | --- |
|  |  |

**3.3创建分组视图**

|  |  |
| --- | --- |
|  |  |

**3.4创建带WITH CHECK OPTION的行列子集视图**

执行已完成，但有错误。

结果: near "with": syntax error

**3.5 查询视图**

**3.6更新视图**

执行已完成，但有错误。

结果: cannot modify IS\_Student because it is a view

**3.7删除视图**

执行完成。

结果: 查询执行成功。耗时 0ms

**四、问题解决**

**4.1 问题一**

如3.4所示，创建带WITH CHECK OPTION的行列子集视图执行失败。

**4.1.1 解决方法**

经过检查，SQL语句并无语法问题。询问老师，发现在sqlite上无法创建带WITH CHECK OPTION选项的视图，而在MySQL上可以执行该操作，可以使用更高版本软件执行创建操作。

**4.2 问题二**

如3.6所示，对行列子集视图执行更新操作失败。

**4.2.1 解决方法**

经过检查，SQL语句并无语法问题。sqlite不支持对视图进行更新操作，可尝试使用更高版本的软件。

**五、实验总结**

在本次实验中，我创建了行列子集视图、带虚拟列的视图和分组视图，尝试创建带WITH CHECK OPTION选项的视图,虽然创建失败，但要明确WITH CHECK OPTION选项的作用：对视图进行UPDATE、INSERT和DELETE操作时要保证更新、插入或删除的行满足视图定义中的谓词条件（即子查询的条件表达式）。

与基本表不同，视图是从一个或几个基本表（或视图）导出来的表，是一个虚表，对视图的查询和更新最终要转换为对基本表的查询和更新，即视图消解。数据库中只存放视图的定义，而不存放它对应的数据，它只是用户观察数据库数据的一个窗口。

**实验五**

**一、实验目的与要求**

**1.1实验目的**

1. 使用QT设计建立一个可视化信息管理系统，熟练应用高级语言C++和数据库的SQL语言完善信息化系统的管理；
2. 熟悉并应用数据库的查询，删除，修改，添加等功能；

**1.2实验要求**

1. 设计登陆界面（注意区分教师端和学生端）
2. 教师端：能够完整查阅所有设计的表格的信息，并且对这些表格可以进行单表的增删改操作（可以不考虑表与表之间的级联关系）。设计的表格信息参照课本P79面的三个表，也可以类似于模板的表格设计，但是学生、课程、选修表必须存在。
3. 学生端：可以查询学生本人的全部信息，不可查询其他人的信息，并且学生端只可查询不可修改。比如学生端可以查询到学生个人基于学生、课程、选修三表连接后的全部信息。（模板建立了四个表，但给出的是学生个人信息表、选修表、奖惩信息表的连接结果，本次需要完成学生个人基于所有建立的表格连接后的全部信息呈现）
4. 在上述基础上可以增加附加的功能，如·注册功能、界面美化、多表级联等。

**二、实验步骤**

1. 安装Qt并且配置支持SQL语句的环境(使用Sqlite进行配置)；
2. 在上述步骤基础上，创建可视化界面和按钮操作；
3. 在Qt中进行编程，设计一个完整含界面开发的信息查询系统，系统登录端分为教师和学生两部分；实现对学生信息、课程信息、学生选修及成绩信息的管理；
4. 运行系统，检测系统功能的完整性和准确性。

**三、需求分析**

**3.1表**

在学生信息管理系统中需要建立五张表：

（1）学生信息表

|  |  |
| --- | --- |
| 属性名 | 说明 |
| Sno | 学生学号 |
| Sname | 学生姓名 |
| Ssex | 学生性别 |
| Sage | 学生年龄 |
| Sdept | 学生系别 |

（2）课程信息表

|  |  |
| --- | --- |
| 属性名 | 说明 |
| Cno | 课程号 |
| Cname | 课程名 |
| Cpno | 先行课号 |
| Ccredit | 课程学分 |

（3）选修关系表

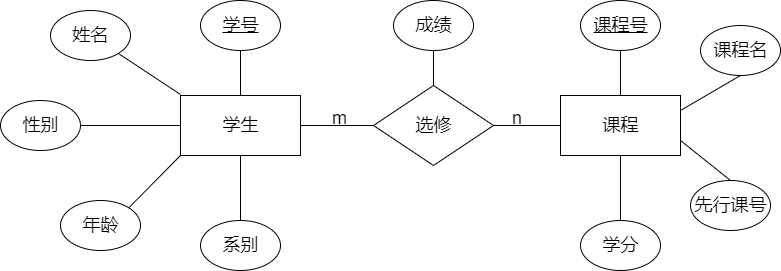
|  |  |
| --- | --- |
| 属性名 | 说明 |
| Sno | 学生学号 |
| Cno | 课程 |
| Grade | 分数 |

（4）学生密码表

|  |  |
| --- | --- |
| 属性名 | 说明 |
| Sno | 学生学号 |
| Spassword | 学生密码 |

（5）教师密码表

|  |  |
| --- | --- |
| 属性名 | 说明 |
| Tno | 教师工号 |
| Tpassword | 教师密码 |

**3.2整体E-R图**

**3.3关系模型**

E-R图中有两个实体：学生和课程，一个关系：选修关系

学生：Student(Sno,Sname,Ssex,Sage,Sdept)

课程：Course(Cno,Cname,Cpno,Ccredit)

选修：SC(Sno,Cno,Grade)

**3.4权限设计**

（1）学生端：只能查看学生的本人的信息，不能查看其他人信息，且不具备添加、修改和删除信息的功能，但可以修改本人的登录密码。

（2）教师端：可以查看学生的全部信息，能够对学生信息表、课程信息表和选修关系表进行增加、删除和修改操作，可以查看并修改学生密码表，对学生的登录权限进行限制，且可以修改本账号的密码。

**3.5功能需求**

1. 登录界面：读取数据库中的学生密码表和教师密码表，对学生学号和教 师工号进行判别，进入不同的学生信息管理系统客户端；
2. 学生端界面：拥有本人所有信息查看功能和修改密码功能，且可返回登 录界面；
3. 教师端界面：分为五个视图界面，分别显示为学生全部信息、学生信息 表、课程信息表、选修关系表和学生密码表，对其拥有不同的操作，也可 进行密码修改操做或返回登录界面；
4. 密码修改界面：分为三个输入框：旧密码、新密码和确认新密码；
5. 属性的域：如学号只能为“2020”开头，性别只能在“男”和“女” 中选择，对年龄要有范围限制等。

**四、功能设计**

**4.1登录部分**

（1）设计思想

* 连接数据库，记录账号和密码输入框的输入内容；
* 检索教师密码表，如果匹配，进入教师端界面；
* 检索学生密码表，如果匹配，进入学生端界面；
* 检索失败，账号和密码清空，弹出警告框

（2）代码部分

void page\_login::**on\_btn\_login\_clicked**()

{

//数据库查找用户名和密码

myconnect1();

name=ui->le\_user->text();

pwd=ui->le\_password->text();

QSqlTableModel model;

model.*setTable*("T\_password");

model.*setFilter*(tr("Tno = '%1' and Tpassword = '%2'").arg(name).arg(pwd));

model.*select*();

if(model.*rowCount*()==1){

QMessageBox::information(this, tr("提示"), tr("登录成功"));

hide();

manage m;

m.show();

m.*exec*();

}

else{

model.*setTable*("password");

model.*setFilter*(tr("Sno = '%1' and pwd = '%2'").arg(name).arg(pwd));

model.*select*();

if(model.*rowCount*()==1) //成功进入主界面

{

QMessageBox::information(this, tr("提示"), tr("登录成功"));

*accept*();

}

else //失败可以重新输入

{

QMessageBox::warning(this, tr("warn"), tr("用户名或者密码不正确"));

ui->le\_user->clear();

ui->le\_password->clear();

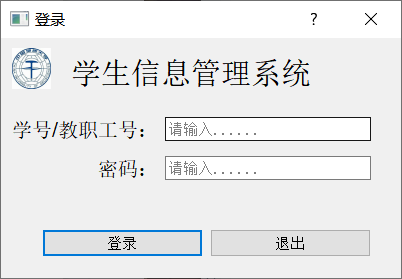
ui->le\_user->setFocus();

}

}

}

（3）界面设计



**4.2 学生端部分**

（1）设计思想

* 初始化QSqlQuery函数
* 编写查询学生全部信息的SQL语句
* 更改显示的列名
* 将结果显示在界面

（2）代码部分

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

studentqrModel=new QSqlQueryModel(this);//只读指针

QString num=suser;

QSqlQuery qr;

qr.prepare("select Student.Sno,Sname,Ssex,Sage,Sdept,Course.Cno,Cname,Grade,Ccredit,Cpno from Student,SC,Course where Student.Sno=SC.Sno and SC.Cno=Course.Cno and Student.Sno=?");//?为预留的用户输入的东西

qr.addBindValue(num);//可以多个

qr.exec();

studentqrModel->setQuery(qr);

studentqrModel->*setHeaderData*(0,Qt::Horizontal,tr("学号"));

studentqrModel->*setHeaderData*(1,Qt::Horizontal,tr("姓名"));

studentqrModel->*setHeaderData*(2,Qt::Horizontal,tr("性别"));

studentqrModel->*setHeaderData*(3,Qt::Horizontal,tr("年龄"));

studentqrModel->*setHeaderData*(4,Qt::Horizontal,tr("系"));

studentqrModel->*setHeaderData*(5,Qt::Horizontal,tr("课程号"));

studentqrModel->*setHeaderData*(6,Qt::Horizontal,tr("课程名"));

studentqrModel->*setHeaderData*(7,Qt::Horizontal,tr("成绩"));

studentqrModel->*setHeaderData*(8,Qt::Horizontal,tr("学分"));

studentqrModel->*setHeaderData*(9,Qt::Horizontal,tr("先行课程号"));

ui->tableView->*setModel*(studentqrModel);

}

1. 界面设计

**4.3教师端部分**

**4.3.1查询**

（1）设计思想

* 显示查询框
* 判断对应属性是否为空，不为空即作为条件
* 根据条件查询
* 修改显示的列名
* 显示查询结果

（2）代码部分

void manage::on\_select\_s\_clicked()

{

student s;

s.show();

s.exec();

//model1=new QSqlTableModel(this);

model1->setTable("Student");

if(!sno.isEmpty()){

model1->setFilter(QObject::tr("Sno = '%1'").arg(sno));

}

if(!sname.isEmpty()){

model1->setFilter(QObject::tr("Sname = '%1'").arg(sname));

}

if(!ssex.isEmpty()){

model1->setFilter(QObject::tr("Ssex = '%1'").arg(ssex));

}

if(!sage.isEmpty()){

model1->setFilter(QObject::tr("Sage = '%1'").arg(sage));

}

if(!sdept.isEmpty()){

model1->setFilter(QObject::tr("Sdept = '%1'").arg(sdept));

}

model1->select(); //显示结果

model1->setHeaderData(0, Qt::Horizontal, tr("学号"));

model1->setHeaderData(1, Qt::Horizontal, tr("姓名"));

model1->setHeaderData(2, Qt::Horizontal, tr("性别"));

model1->setHeaderData(3, Qt::Horizontal, tr("年龄"));

model1->setHeaderData(4, Qt::Horizontal, tr("系"));

ui->tableView\_1->setModel(model1);

}

**4.3.2修改**

1. 设计思想

* 选中要修改的单元格
* 修改单元格内容
* 点击“保存”，手动提交（此时会对各个属性的限制条件进行判断）
* 显示修改结果

（2）代码部分

注：由于QSqlTableModel模型直接可以在表格中修改，此处只提供手动提交代码

void manage::on\_save\_s\_clicked()

{

model1->database().transaction();//开始事务

if (model1->submitAll()) {//提交事务

model1->database().commit();//确认结束事务

} else {

model1->database().rollback();//回滚结束事务

QMessageBox::warning(this, tr("Cached Table"),

tr("The database reported an error: %1")

.arg(model1->lastError().text()));

}

}

**4.3.3增加**

（1）设计思想

* 获得表的行数
* 在原表后增加一行
* 填写增加的信息
* 点击保存，手动提交（此时会对各个属性的限制条件进行判断）
* 显示添加结果

（2）代码部分

注：此处代码仅为添加表格行的代码

void manage::on\_insert\_s\_clicked()

{

model1->setTable("Student");

int rowNum = model1->rowCount(); //获得表的行数

int id = rowNum+1;

model1->insertRow(rowNum);

model1->setData(model1->index(rowNum,0),"必须填写"); /

model1->submitAll(); //可以直接提交

model1->select(); //显示结果

ui->tableView\_1->setModel(model1);

}

**4.3.4删除**

（1）设计思想

* 检索当前选中的单元格的行号（可选择多行）
* 弹出警告框，确认是否删除
* 若点击“确认”,则对当前选中的行号进行删除；若点击“取消”，则不执行操作
* 显示删除后的结果

（2）代码部分

void manage::**on\_delete\_s\_clicked**()

{

model1=new QSqlTableModel(this);

model1->*setTable*("Student");

int row = ui->tableView\_1->currentIndex().row(); model1->*select*(); //显示结果

QItemSelectionModel \*selections = ui->tableView\_1->selectionModel(); //返回当前的选择模式

QModelIndexList selecteds = selections->selectedIndexes(); 、 //返回所有选定的模型项目索引列表

int ok = QMessageBox::warning(this,tr("删除选中的行!"),tr("你确定删除当前选取中的行吗?"),QMessageBox::Yes,QMessageBox::No);

if(ok == QMessageBox::Yes)

{

foreach (QModelIndex index, selecteds)

{

int curRow = index.row(); //删除所有被选中的行

model1->removeRow(curRow);

}

model1->submitAll(); //提交，在数据库中删除该行

}

else

{

model1->revertAll(); //如果不删除，则撤销

}

}

**4.3.5刷新**

（1）设计思想

* 重新查询数据库中的表
* 显示查询结果

（2）代码部分

void manage::on\_flushed\_s\_clicked()

{

//model1=new QSqlTableModel(this);

model1->setTable("Student");

model1->select(); //显示结果

model1->setHeaderData(0, Qt::Horizontal, tr("学号"));

model1->setHeaderData(1, Qt::Horizontal, tr("姓名"));

model1->setHeaderData(2, Qt::Horizontal, tr("性别"));

model1->setHeaderData(3, Qt::Horizontal, tr("年龄"));

model1->setHeaderData(4, Qt::Horizontal, tr("系"));

ui->tableView\_1->setModel(model1);

}

**4.3.6撤回**

（1）设计思想

在表格中增加或修改的内容，未保存之前，可以回退

（2）代码部分

void manage::on\_return\_s\_clicked()

{

model1->revertAll();//撤回模型的更改\*/

}

**4.3.7升序**

（1）设计思想

按某一属性实现显示结果的升序排列

（2）代码部分

void manage::on\_pushButton\_10\_clicked()

{

model1->setTable("Student");

model1->setSort(0,Qt::AscendingOrder);

model1->select(); //显示结果

ui->tableView\_1->setModel(model1);

}

**4.3.8降序**

（1）设计思想

按某一属性实现显示结果的降序排列

（2）代码部分

void manage::on\_pushButton\_11\_clicked()

{

model1->setTable("Student");

model1->setSort(0,Qt::DescendingOrder);

model1->select(); //显示结果

ui->tableView\_1->setModel(model1);

}

**4.3.9界面设计（部分）**

**4.4密码修改部分**

（1）设计思想

* 输入旧密码、新密码和再次输入新密码新密码
* 判断旧密码是否输入错误
* 判断新密码是否一致，不一致弹出警告框，将新密码文本框清空
* 判断新旧密码是否一致，一致弹出警告框，将新密码文本框清空
* 若无上述情况，则修改成功

（2）代码部分

void password::**on\_pushButton\_clicked**()

{ QString pwd=ui->lineEdit\_1->text();

QString npwd1=ui->lineEdit\_2->text();

QString npwd2=ui->lineEdit\_3->text();

if(pwd!=spwd){

QMessageBox::warning(this, tr("warn"), tr("旧密码输入错误"));

ui->lineEdit\_1->clear();

ui->lineEdit\_1->setFocus();

}

else if(npwd1!=npwd2){

QMessageBox::warning(this, tr("warn"), tr("新密码不一致"));

ui->lineEdit\_3->clear();

ui->lineEdit\_3->setFocus();

}

else if(npwd1==spwd){

QMessageBox::warning(this, tr("warn"), tr("新旧密码不能完全一致"));

ui->lineEdit\_2->clear();

ui->lineEdit\_3->clear();

ui->lineEdit\_2->setFocus();

}

else {

QMessageBox::information(this, tr("提示"), tr("修改成功"));

Spwd(npwd1);

this->hide();

}

}

**五、效果呈现**

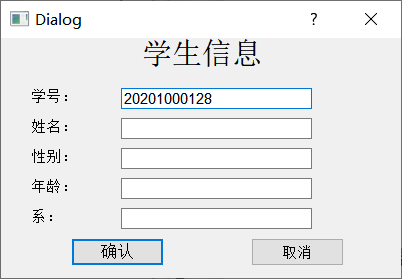
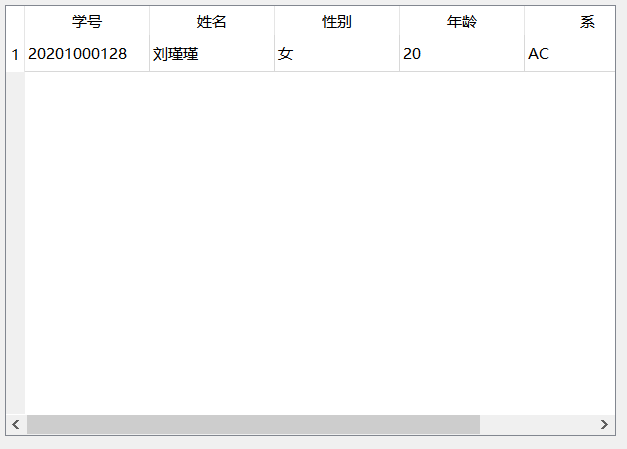
**5.1登录**

|  |  |
| --- | --- |
| （1）登录成功 | （2）登陆失败 |
|  |  |

**5.2学生端**

**5.3教师端（以学生表为例）**

1. 查询

* 查询框中输入
* 查询结果

1. 添加

|  |  |
| --- | --- |
| 添加前 | 添加后 |
|  |  |
| 检查主属性唯一 | 检查限制条件 |
|  |  |

1. 删除

|  |  |
| --- | --- |
| 删除前 | 删除后 |
|  |  |

1. 修改

|  |  |
| --- | --- |
| 修改前 | 修改后 |
|  |  |

1. 排列顺序

|  |  |
| --- | --- |
| 升序 | 降序 |
|  |  |

1. 学生登录管理

在该界面可对学生的账号和密码进行管理，如果允许学生登录，则保留其账号和密码，若不允许登录，则直接删除该学生的账号和密码，也可添加新的学生账号，或者直接修改学生密码。

1. 全部信息显示

**5.4密码界面**

|  |  |
| --- | --- |
| 判断旧密码是否输入错误（旧密码：123456） | |
|  |  |
| 判断新密码是否一致 | |
|  |  |
| 判断新旧密码是否一致 | |
|  |  |
| 修改成功 | |
|  |  |

**六、问题解决**

**6.1问题一**

（1）问题一：执行删除操作时，选中元组，点击“确定”后，删除操作无法执行，仍然显示原数据。

（2）解决方法：将select()函数放在remove()函数之前即可执行操作；

**6.2问题二**

（1）问题二：无法判断输入的查询条件时属于哪一个属性

（2）解决方法：判断lineEdit中内容是否为空，不为空则包含查询的条件，使用setFilter()函数设置查询条件，select()进行查询；

**6.3问题三**

（1）问题三：不能限制条件，如学号为“2020”开头，性别为“男”和“女”

（2）解决方法：在表的定义中使用CHECK增加约束条件，QSqlTableModel模型可以实现检查自动检查，比如性别是"男"或“女”，可以限制年龄范围，或者限制学号的范围，不符合约束条件会自动弹出警告框，提交操作失败。

**七、实验总结**

在本次综合实验的过程中，我学习到了如何在QT中与建立好的数据库建立连接，学习了QSqlQuery的使用，尝试使用了与SQL相关的两种模型：QSqlTableModel和QSqlQueryModel进行数据库相关操作。

QSqlQuery能执行任意SQL语句并获得执行结果，如select、insert、update和delete,可以直接编写SQL语句进行相关操作，可以锻炼我们对SQL语言的应用能力，缺点是；许多功能需要我们自己编写，集成度低。QSqlQueryModel是一个只读数据模型，只能进行读操作，不可以编辑数据，但它可以防止他人修改，保证了数据库数据的安全性，一般与QSqlQuery一起使用；而QSqlTableModel是单表模型，不需要书写SQL语句，利用封装好的函数可以实现对数据的编辑、插入和删除等操作，并实现数据的排序和过滤，在提交修改数据库内容时会自动进行约束条件的检查，使用相对简单，缺点是只能对单个数据表表进行操作，查询显示的数据量不超过256。

在熟练使用两种模型的基础上，我完善了学生信息管理系统的界面设计和功能设计，对QT信号与槽机制有了更深刻的理解，对于QDialog和Qwidget两类设计界面的使用也更加熟练。

另外，我也意识到要做好一个功能完善的信息管理系统是很困难的，在设计系统的时候，由于我是设计者，我总会下意识的认为输入的内容一定符合表的定义，自动忽略容易出错的地方，但在实际应用过程中，使用者往往不是设计者，可能会出现一些意想不到的情况，这就要求我们在设计过程中要充分考虑约束条件。

在这段时间中，我遇到了很多困难，大部分通过上网查资料得到了解决，比较困难的问题则向老师和同学请教，在此诚挚地感谢王老师和闫学长的帮助。比较遗憾的是，由于时间关系没能考虑多表级联的问题，没有充分考虑数据库的实体完整性、参照完整性和用户定义完整性的约束问题，可能这些问题会很复杂，但也更具挑战性，如果以后有机会，我一定仔细考虑这些问题。