

华中科技大学

数据库系统原理实践报告

综合设计题目： 成绩管理系统

姓 名： 胡思勖

学 院： 计算机科学与技术学院

专 业： 计算机科学与技术

班 级： 计卓 1501

学 号： U201514898

指导教师： 谢美意

分数	
教师	

2018 年 06 月 29 日

目 录

1 课程任务概述.....	1
2 SQL 实验.....	2
2.1 任务要求.....	2
2.2 完成过程.....	2
2.3 任务总结.....	13
3 安全性及完整性实验.....	14
3.1 任务要求	14
3.2 完成过程	14
3.3 任务总结.....	24
4 综合实践任务.....	25
4.1 系统设计目标.....	25
4.2 需求分析.....	25
4.3 总体设计.....	26
4.4 数据库设计.....	26
4.5 详细设计与实现.....	32
4.6 系统测试.....	41
4.7 系统设计与实现总结.....	45
5 课程总结.....	46
附录	47

1 课程任务概述

本次数据库系统原理课程实验的任务共有 3 个，前两个任务为对于数据库基础的认知实验，最后一个是一个综合实践。3 个实验的任务难度逐级递增，在前两个实验打下良好基础的情况下最终实现一个 C/S 模式的成绩管理系统。

实验任务一：SQL 实验

熟悉 DBMS 软件的安装以及使用，并且能够使用 SQL 语言进行表的创建、添加、删除、修改、查询等操作。对于查询的要求较高。此外还需要对于数据库事务有一定的了解，能够了解在不同的事务隔离级别进行数据库操作的异同。

实验任务二：安全性及完整性实验

在数据库的不同表格中加上不同的安全性和完整性控制，使用 DBMS 的权限管理功能实现这些控制，并对存储过程具有一定的了解和认识。

实验任务三：综合实践任务

选定一个数据库应用的题目，并在题目限定的应用场景下完成需求分析，数据库设计和应用程序设计，并提交相应的文档。本实验选定的场景为采用 B/S 或 C/S 模式实现一个成绩管理系统。完成课程、学生、教师、选课、授课、成绩等信息的管理。其要求如下：

- 1) 提供管理员、学生、教师三类用户的录入、查询界面；
- 2) 每门课程可以有多次作业或测试，课程总成绩由各项成绩加权累加计算得到，作业或测试的数量不能由系统预先指定，而应由教师本人指定自己所负责课程的作业及测试的数量和各自权重；
- 3) 系统应支持等级评定，允许对不同的级别指定分隔点；
- 4) 提供学生、课程等不同角度的成绩统计功能。

2 SQL 实验

2.1 任务要求

熟悉一种 DBMS 的安装以及使用，并使用此 DBMS 完成一系列的创建、插入、删除、修改、查询等基本操作。对于特殊的查询以及事务的特性有一定的了解并能够使用。

2.2 完成过程

2.2.1 数据定义

1. GOODS: 商品表（商品名称，商品类型）

说明：主码为商品名称；商品类型为电器、文具、服装……

2. PLAZA: 商场表（商场名称，所在地区）

说明：主码为商场名称；所在地区为汉口、汉阳、武昌……

3. SALE: 销售价格表（商品名称，商场名称，价格，促销类型）

说明：主码为（商品名称、商场名称）；促销类型为送券、打折，也可为空值，表示当前未举办任何活动；表中记录如（‘哈森皮靴’，‘大洋百货’，300，‘打折’），同一商场针对不同的商品可能采取不同的促销活动。

创建表格的 SQL 语句如下：

```
create table if not exists GOODS (
    商品名称 nvarchar(10) not null primary key,
    商品类型 nvarchar(5) not null
);

create table if not exists PLAZA (
    商场名称 nvarchar(10) not null primary key,
    所在地区 nvarchar(5) not null
);

create table if not exists SALE (
    商品名称 nvarchar(10) not null,
    商场名称 nvarchar(10) not null,
    价格      decimal(8,2) not null default 0.0,
    促销类型 nvarchar(10),
    primary key (商品名称, 商场名称)
);
```

上述语句分别创建 3 个表格：GOODS、PLAZA 以及 SALE。除了 SALE 中的价格外其余的码类型均为 nvarchar 类型，以更好的支持 Unicode 编码。成功创建表格后使用 show tables 命令显示所有表格，其结果如图 2.1 所示，从图中可以看出，3 个表格均已成功创建。

```

MariaDB [db_lab1]> show tables;
+-----+
| Tables_in_db_lab1 |
+-----+
| GOODS           |
| PLAZA           |
| SALE            |
+-----+
3 rows in set (0.00 sec)

MariaDB [db_lab1]> █

```

图 2.1 表格创建后使用 show tables 的输出

2.2.2 数据更新

(1) 用 SQL 语句完成三个关系的数据插入

对于 GOODS 表的插入，其 SQL 语句如下，对于另外两个表的插入语句类似，由于篇幅原因不再给出。

```

insert into GOODS (商品名称, 商品类型) values
('创维*电视',      '电器'),
('格力*空调',      '电器'),
('海尔*冰箱',      '电器'),
('小天鹅*洗衣机', '电器'),
('老人头*t恤',    '服装'),
('LEE*牛仔裤',     '服装'),
('哈森*皮靴',     '服装'),
('七匹狼*夹克',   '服装'),
('晨光*中性笔',   '文具'),
('得力*笔记本',   '文具'),
('老干妈*辣酱',   '食品'),
('奥利奥*饼干',   '食品'),
('海天*酱油',      '食品');

```

插入后，使用 SELECT 语句查看表中的内容，其结果如图 2.2、图 2.3 和图 2.4 所示。从图中可以看出插入语句正常工作。

```

MariaDB [db_lab1]> select * from PLAZA;
+-----+-----+
| 商场名称        | 所在地区 |
+-----+-----+
| 大洋百货        | 武昌    |
| 新世界百货      | 汉口    |
| 校园超市        | 武昌    |
| 汉商购物中心    | 汉阳    |
| 王府井百货      | 汉口    |
| 群光广场        | 武昌    |
+-----+-----+
6 rows in set (0.00 sec)

MariaDB [db_lab1]> █

```

图 2.2 使用 SELET 语句查看 PLAZA 表中的内容

```
MariaDB [db_lab1]> select * from GOODS;
+-----+-----+
| 商品名称 | 商品类型 |
+-----+-----+
| LEE*牛仔裤 | 服装 |
| 七匹狼*夹克 | 服装 |
| 创维*电视 | 电器 |
| 哈森*皮靴 | 服装 |
| 奥利奥*饼干 | 食品 |
| 小天鹅*洗衣机 | 电器 |
| 得力*笔记本 | 文具 |
| 晨光*中性笔 | 文具 |
| 格力*空调 | 电器 |
| 海天*酱油 | 食品 |
| 海尔*冰箱 | 电器 |
| 老人头*t恤 | 服装 |
| 老干妈*辣酱 | 食品 |
+-----+
13 rows in set (0.00 sec)
```

图 2.3 使用 SELECT 语句查看 GOODS 表中的内容

```
MariaDB [db_lab1]> select * from SALE;
+-----+-----+-----+-----+
| 商品名称 | 商场名称 | 价格 | 促销类型 |
+-----+-----+-----+
| LEE*牛仔裤 | 大洋百货 | 380.00 | 打折 |
| LEE*牛仔裤 | 汉商购物中心 | 500.00 | 送券 |
| 七匹狼*夹克 | 新世界百货 | 880.00 | 打折 |
| 创维*电视 | 群光广场 | 3188.00 | 打折 |
| 哈森*皮靴 | 大洋百货 | 300.00 | 打折 |
| 奥利奥*饼干 | 校园超市 | 8.50 | NULL |
| 奥利奥*饼干 | 汉商购物中心 | 8.00 | NULL |
| 小天鹅*洗衣机 | 大洋百货 | 2288.00 | NULL |
| 得力*笔记本 | 校园超市 | 4.50 | NULL |
| 得力*笔记本 | 汉商购物中心 | 5.00 | NULL |
| 晨光*中性笔 | 校园超市 | 1.00 | NULL |
| 晨光*中性笔 | 汉商购物中心 | 1.50 | NULL |
| 格力*空调 | 大洋百货 | 4388.00 | 打折 |
| 格力*空调 | 群光广场 | 4588.00 | 打折 |
| 海尔*冰箱 | 汉商购物中心 | 3888.00 | NULL |
| 老人头*t恤 | 大洋百货 | 1200.00 | 送券 |
| 老干妈*辣酱 | 大洋百货 | 9.00 | NULL |
+-----+
17 rows in set (0.00 sec)
```

图 2.4 使用 SELECT 语句查看 SALE 中的内容

(2) 将 SALE 表中活动类型为打折的记录插入到新表 SALE_CHEAP 中

使用 CREATE 语句结合 SELECT 语句创建一个新表，其 SQL 语句如下：

```
create table if not exists SALE_CHEAP as (
  select * from SALE
  where 促销类型="打折"
);
```

然后使用 SELECT 语句查看 SALE_CHEAP 中的内容，结果如图 2.5 所示

```
MariaDB [db_lab1]> select * from SALE_CHEAP;
+-----+-----+-----+-----+
| 商品名称 | 商场名称 | 价格 | 促销类型 |
+-----+-----+-----+-----+
| LEE*牛仔裤 | 大洋百货 | 380.00 | 打折
| 七匹狼*夹克 | 新世界百货 | 880.00 | 打折
| 创维*电视 | 群光广场 | 3188.00 | 打折
| 哈森*皮靴 | 大洋百货 | 300.00 | 打折
| 格力*空调 | 大洋百货 | 4388.00 | 打折
| 格力*空调 | 群光广场 | 4588.00 | 打折
+-----+-----+-----+-----+
6 rows in set (0.00 sec)

MariaDB [db_lab1]>
```

图 2.5 SELECT 语句查看 SALE_CHEAP 中的内容

(3) 群光广场的创维电视停止打折活动，价格恢复为 3988，对于 SALE 表和 SALE_CHEAP 表做相应的修改；

使用一条 update 语句和一条 delete 语句进行修改

```
update SALE
set 价格=3988, 促销类型=null
where 商场名称='群光广场' and 商品名称='创维*电视';

delete from SALE_CHEAP
where 商场名称='群光广场' and 商品名称='创维*电视';
```

修改后的 SALE 和 SALE_CHEAP 中的内容如图 2.6 和图 2.7 所示，可以看出条目已被正确修改。

```
MariaDB [db_lab1]> select * from SALE;
+-----+-----+-----+-----+
| 商品名称 | 商场名称 | 价格 | 促销类型 |
+-----+-----+-----+-----+
| LEE*牛仔裤 | 大洋百货 | 380.00 | 打折
| LEE*牛仔裤 | 汉商购物中心 | 500.00 | 送券
| 七匹狼*夹克 | 新世界百货 | 880.00 | 打折
| 创维*电视 | 群光广场 | 3988.00 | NULL
| 哈森*皮靴 | 大洋百货 | 300.00 | 打折
| 奥利奥*饼干 | 校园超市 | 8.50 | NULL
| 奥利奥*饼干 | 汉商购物中心 | 8.00 | NULL
| 小天鹅*洗衣机 | 大洋百货 | 2288.00 | NULL
| 得力*笔记本 | 校园超市 | 4.50 | NULL
| 得力*笔记本 | 汉商购物中心 | 5.00 | NULL
| 晨光*中性笔 | 校园超市 | 1.00 | NULL
| 晨光*中性笔 | 汉商购物中心 | 1.50 | NULL
| 格力*空调 | 大洋百货 | 4388.00 | 打折
| 格力*空调 | 群光广场 | 4588.00 | 打折
| 海尔*冰箱 | 汉商购物中心 | 3888.00 | NULL
| 老人头*t恤 | 大洋百货 | 1200.00 | 送券
| 老干妈*辣酱 | 大洋百货 | 9.00 | NULL
+-----+-----+-----+-----+
17 rows in set (0.00 sec)
```

图 2.6 修改后的 SALE 表

```

MariaDB [db_lab1]> select * from SALE_CHEAP;
+-----+-----+-----+-----+
| 商品名称 | 商场名称 | 价格 | 促销类型 |
+-----+-----+-----+-----+
| LEE*牛仔裤 | 大洋百货 | 380.00 | 打折 |
| 七匹狼*夹克 | 新世界百货 | 880.00 | 打折 |
| 哈森*皮靴 | 大洋百货 | 300.00 | 打折 |
| 格力*空调 | 大洋百货 | 4388.00 | 打折 |
| 格力*空调 | 群光广场 | 4588.00 | 打折 |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

MariaDB [db_lab1]> █

```

图 2.7 修改后的 SALE_CHEAP 表

(4) 基于 SALE_CHEAP 表创建一个统计每个打折商品平均价格的视图。

使用 CREATE VIEW 创建视图，具体语句如下：

```

create view if not exists SALE_VIEW as (
    select 商品名称, AVG(价格)
    from SALE_CHEAP
    group by 商品名称
);

```

创建视图后使用 SELECT 查看视图的内容，如图 2.8 所示，可以看到视图已被正确创建。

```

MariaDB [db_lab1]> select * from SALE_VIEW;
+-----+-----+
| 商品名称 | AVG(价格) |
+-----+-----+
| LEE*牛仔裤 | 380.000000 |
| 七匹狼*夹克 | 880.000000 |
| 哈森*皮靴 | 300.000000 |
| 格力*空调 | 4488.000000 |
+-----+-----+
4 rows in set (0.00 sec)

MariaDB [db_lab1]> █

```

图 2.8 创建后的视图

2.2.3 数据查询

(1) 查询所有没有任何促销活动的商品及其所在的商场，结果按照商品名排序；

```

-----
select 商品名称, 商场名称
from SALE
where 促销类型 is null
order by 商品名称
-----

+-----+-----+
| 商品名称 | 商场名称 |
+-----+-----+
| 创维*电视 | 群光广场 |
| 奥利奥*饼干 | 校园超市 |
| 奥利奥*饼干 | 汉商购物中心 |
| 小天鹅*洗衣机 | 大洋百货 |
| 得力*笔记本 | 校园超市 |
| 得力*笔记本 | 汉商购物中心 |
| 晨光*中性笔 | 校园超市 |
| 晨光*中性笔 | 汉商购物中心 |
| 海尔*冰箱 | 汉商购物中心 |
| 老干妈*辣酱 | 大洋百货 |
+-----+-----+
10 rows in set (0.000 sec)

```

图 2.9 使用 is null 和 order by 进行查询

语句与查询结果如图 2.9 所示，由于在启动数据库客户端时打开了 verbose mode，因此会显示执行的语句，此处不再单独给出 SQL 语句，下同。这一步需要注意的是不能使用= null 而要使用 is null 来过滤。

- (2) 查询价格在 200~500 元之间的商品名称、所在的商场名称、价格，结果按照商场名称排序；

语句和结果如图图 2.10 所示。使用 and 将两个查询条件进行连接，并使用 order by 对于查询结果进行排序。

```
-----  
select 商品名称, 商场名称, 价格  
from SALE  
where 价格 >= 200 and 价格 <= 500  
order by 商场名称  
-----  
  
+-----+-----+-----+  
| 商品名称 | 商场名称 | 价格 |  
+-----+-----+-----+  
| LEE*牛仔裤 | 大洋百货 | 380.00 |  
| 哈森*皮靴 | 大洋百货 | 300.00 |  
| LEE*牛仔裤 | 汉商购物中心 | 500.00 |  
+-----+-----+-----+  
3 rows in set (0.001 sec)
```

图 2.10 使用 where 和 order by 查询结果

- (3) 查询每种商品的商品名称、最低售价、最高售价；

查询语句与结果如图图 2.11 所示，使用 join 进行自然连接后再使用集函数查询最大和最小值

```
-----  
select 商品类型, min(价格), max(价格)  
from GOODS join SALE  
on GOODS.商品名称=SALE.商品名称  
group by 商品类型  
-----  
  
+-----+-----+-----+  
| 商品类型 | min(价格) | max(价格) |  
+-----+-----+-----+  
| 文具 | 1.00 | 5.00 |  
| 服装 | 300.00 | 1200.00 |  
| 电器 | 2288.00 | 4588.00 |  
| 食品 | 8.00 | 9.00 |  
+-----+-----+-----+  
4 rows in set (0.001 sec)
```

图 2.11 使用 join 进行组合后再查询

- (4) 查询以“打折”方式销售的商品总数超过 2 种的商场名称及其所在地区；

```
-----  
select PLAZA.商场名称, PLAZA.所在地区  
from PLAZA join SALE  
on PLAZA.商场名称=SALE.商场名称  
where 促销类型='打折'  
group by PLAZA.商场名称  
having count(*)>2  
-----  
  
+-----+-----+  
| 商场名称 | 所在地区 |  
+-----+-----+  
| 大洋百货 | 武昌 |  
+-----+-----+  
1 row in set (0.001 sec)
```

图 2.12 使用 having 语句的查询结果

查询结果如图 2.12 所示，此时使用了 having 对于组进行过滤。

(5) 查询以“老”字开头的所有商品的名称;

查询语句与结果如图 2.13, 需要注意的是需要使用 like 过滤条件进行字符串匹配。

```
-----  
select 商品名称  
from GOODS  
where 商品名称 like '老%'  
-----  
  
+-----+  
| 商品名称 |  
+-----+  
| 老人头*t恤 |  
| 老干妈*辣酱 |  
+-----+  
2 rows in set (0.000 sec)
```

图 2.13 使用 like 过滤进行查询的结果

(6) 查询同时销售“晨光*中性笔”和“得力*笔记本”的商场名称;

如图图 2.14 所示, 使用 intersect 求两个查询结果的交集。再较老的数据库上这一语句可能失败。使用了较高版本的 Mariadb 后本条语句执行成功。

```
-----  
(select 商场名称  
from SALE  
where 商品名称='晨光*中性笔'  
) intersect (select 商场名称  
from SALE  
where 商品名称='得力*笔记本'  
)  
-----  
  
+-----+  
| 商场名称 |  
+-----+  
| 校园超市 |  
| 汉商购物中心 |  
+-----+  
2 rows in set (0.001 sec)
```

图 2.14 求两个查询的交集

(7) 查询未举办任何活动的商场;

查询语句与结果如图 2.15 所示。需要注意再 where 子句中是不能使用 COUNT 函数的, 需要使用 having 对组进行过滤。

```
-----  
select 商场名称  
from SALE  
group by 商场名称  
having COUNT(促销类型) = 0  
-----  
  
+-----+  
| 商场名称 |  
+-----+  
| 校园超市 |  
+-----+  
1 row in set (0.001 sec)
```

图 2.15 再 having 中使用集函数进行查询

(8) 查询出售商品种类最多的商场名称；

如图 2.16 所示，对这一语句需要使用嵌套查询，才能将商品种类与所有其他的商品种类进行对比。

```
-----  
select 商场名称  
from SALE  
group by 商场名称  
having count(*) >= all(  
    select count(*)  
    from SALE  
    group by 商场名称  
)  
-----  
+-----+  
| 商场名称 |  
+-----+  
| 大洋百货 |  
+-----+  
1 row in set (0.001 sec)
```

图 2.16 使用嵌套查询

(9) 查询出售商品类型最多的商场名称；

如图 2.17 所示由于既要有类型也要有商场名称，因此需要连接，又要求商品数量最多的，因此需要嵌套查询。

```
-----  
select 商场名称  
from  
SALE inner join GOODS  
on SALE.商品名称 = GOODS.商品名称  
group by SALE.商场名称  
having count(distinct GOODS.商品类型) >= all(  
    select count(distinct GOODS.商品类型)  
    from SALE inner join GOODS  
    on SALE.商品名称 = GOODS.商品名称  
    group by SALE.商场名称  
)  
-----  
+-----+  
| 商场名称 |  
+-----+  
| 汉商购物中心 |  
+-----+  
1 row in set (0.001 sec)
```

图 2.17 复杂的嵌套查询

(10) 查询所销售的商品包含了“校园超市”所销售的所有商品的商场名称。

```
-----  
select oldSALE.商场名称  
from (  
    select * from SALE  
) as oldSALE  
inner join (  
    select 商品名称 from SALE  
    where 商场名称 = '校园超市'  
) as newSALE  
on oldSALE.商品名称 = newSALE.商品名称  
group by oldSALE.商场名称  
having  
    count(distinct oldSALE.商品名称) = count(distinct newSALE.商品名称)  
-----  
+-----+  
| 商场名称 |  
+-----+  
| 校园超市 |  
| 汉商购物中心 |  
+-----+  
2 rows in set (0.001 sec)
```

图 2.18 复杂的嵌套查询

查询语句以及结果如图 2.18 所示。对于 having 子句中的条件需要注意要去重，否则不能得出正确的答案。

(11) 查询所有商品的名称及出售该商品的商场，要求未在任何商场出售的商品名称也能显示出来。

查询语句及结果如图图 2.19 所示。由于需要未在商场出售的商品也显示，因此需要 outer join 来包含不能匹配商场的商品名称。

```
select GOODS.商品名称, 商场名称
from GOODS left outer join SALE
on GOODS.商品名称 = SALE.商品名称
-----  
+-----+-----+
| 商品名称 | 商场名称 |
+-----+-----+
| LEE*牛仔裤 | 大洋百货 |
| LEE*牛仔裤 | 汉商购物中心 |
| 七匹狼*夹克 | 新世界百货 |
| 创维*电视 | 群光广场 |
| 哈森*皮靴 | 大洋百货 |
| 奥利奥*饼干 | 校园超市 |
| 奥利奥*饼干 | 汉商购物中心 |
| 小天鹅*洗衣机 | 大洋百货 |
| 得力*笔记本 | 校园超市 |
| 得力*笔记本 | 汉商购物中心 |
| 晨光*中性笔 | 校园超市 |
| 晨光*中性笔 | 汉商购物中心 |
| 格力*空调 | 大洋百货 |
| 格力*空调 | 群光广场 |
| 海天*酱油 | NULL |
| 海尔*冰箱 | 汉商购物中心 |
| 老人头*t恤 | 大洋百货 |
| 老干妈*辣酱 | 大洋百货 |
+-----+-----+
18 rows in set (0.001 sec)
```

图 2.19 使用 outer join 进行连接的查询

2.2.4 查询提高（选做）

(1) 在 SALE 表中增加一个活动截止时间，查出所有在整点时刻截止活动的商品；

使用以下语句插入截止时间列。查询用的语句如图图 2.20 所示。由于篇幅过长不给出所有的截止时间。从结果可以看出，语句正常执行。

```
alter table SALE
add 活动截止时间 timestamp;
```

→ 查出所有在整点时刻截止活动的商品

```
select * from SALE
where minute(活动截止时间)=0 and second(活动截止时间)=0
-----  
+-----+-----+-----+-----+-----+
| 商品名称 | 商场名称 | 价格 | 促销类型 | 活动截止时间 |
+-----+-----+-----+-----+-----+
| 奥利奥*饼干 | 汉商购物中心 | 8.00 | NULL | 2018-07-19 08:00:00 |
| 小天鹅*洗衣机 | 大洋百货 | 2288.00 | NULL | 2018-10-26 17:00:00 |
| 得力*笔记本 | 校园超市 | 4.50 | NULL | 2016-02-04 18:00:00 |
| 晨光*中性笔 | 校园超市 | 1.00 | NULL | 2019-12-14 06:00:00 |
| 晨光*中性笔 | 汉商购物中心 | 1.50 | NULL | 2019-05-26 01:00:00 |
| 格力*空调 | 大洋百货 | 4388.00 | 打折 | 2016-10-08 12:00:00 |
| 老人头*t恤 | 大洋百货 | 1200.00 | 送券 | 2018-11-19 05:00:00 |
+-----+-----+-----+-----+-----+
7 rows in set (0.000 sec)
```

图 2.20 整点时刻查询语句

(2) 查询活动截止时间在本月最后一天的 SALE 记录；

查询语句与结果如图 2.21 所示。由于本月最后一天没有记录，因此查询结果为空集。

```
→ 查询活动截止时间在本月最后一天的SALE记录
-----
select *
from SALE
where date(活动截止时间)=last_day(curdate())
-----

Empty set (0.000 sec)
```

图 2.21 月份有关的查询

(3) 查询截止时间在 2016 年的 SALE 记录；

查询语句与结果如图图 2.22 所示，对于年份进行过滤。

```
-----
select *
from SALE
where year(活动截止时间)=2016
-----

+-----+-----+-----+-----+-----+
| 商品名称 | 商场名称 | 价格 | 促销类型 | 活动截止时间 |
+-----+-----+-----+-----+-----+
| LEE*牛仔裤 | 大洋百货 | 380.00 | 打折 | 2016-12-14 12:23:21 |
| 奥利奥*饼干 | 校园超市 | 8.50 | NULL | 2016-08-25 03:06:41 |
| 得力*笔记本 | 校园超市 | 4.50 | NULL | 2016-02-04 18:00:00 |
| 格力*空调 | 大洋百货 | 4388.00 | 打折 | 2016-10-08 12:00:00 |
| 格力*空调 | 群光广场 | 4588.00 | 打折 | 2016-09-12 01:36:41 |
| 老干妈*辣酱 | 大洋百货 | 9.00 | NULL | 2016-12-26 08:50:01 |
+-----+-----+-----+-----+-----+
6 rows in set (0.000 sec)
```

图 2.22 年份有关的查询

(4) 查询本月的最后一天；

查询语句与结果如图 2.23 所示，此查询与数据库无关。

```
-----
select last_day(curdate())
-----

+-----+
| last_day(curdate()) |
+-----+
| 2018-06-30 |
+-----+
1 row in set (0.000 sec)
```

图 2.23 月份与日期有关的查询

(5) 将所有打折商品的活动截止时间推迟一个小时。

更新语句如图 2.24 所示，使用 update 对于所有活动进行更新。由于是所有活动因此无需过滤。

```
-----
update SALE
set `活动截止时间`=timestampadd(hour, 1, `活动截止时间`)
-----

Query OK, 17 rows affected (0.003 sec)
Rows matched: 17  Changed: 17  Warnings: 0
```

图 2.24 推迟活动截止时间

- (6) 将查询结果的某些列的某些值转换成特殊的形式（例如：若商场所在地区为武昌，则在所在地区列中显示“附近”；否则显示“遥远”）

查询语句与结果如图 2.25 所示，需要使用 case 对于结果进行翻译。

```
-----  
select 商场名称,  
case 所在地区  
    when '武昌' then '附近'  
    else '遥远'  
end as '所在地区'  
from PLAZA  
-----  
+-----+-----+  
| 商场名称 | 所在地区 |  
+-----+-----+  
| 大洋百货 | 附近 |  
| 新世界百货 | 遥远 |  
| 校园超市 | 附近 |  
| 汉商购物中心 | 遥远 |  
| 王府井百货 | 遥远 |  
| 群光广场 | 附近 |  
+-----+-----+  
6 rows in set (0.000 sec)
```

图 2.25 需要使用 case 子句对于所选的值进行过滤

- (7) 查询价格个位数为 8 元的商品名称、所在商场名称和价格；

查询语句与结果如图 2.26 所示，取模后对于价格的个位数进行过滤。

```
-----  
select 商品名称, 商场名称, 价格  
from SALE  
where 价格 %10=8  
-----  
+-----+-----+-----+  
| 商品名称 | 商场名称 | 价格 |  
+-----+-----+-----+  
| 创维*电视 | 群光广场 | 3988.00 |  
| 奥利奥*饼干 | 汉商购物中心 | 8.00 |  
| 小天鹅*洗衣机 | 大洋百货 | 2288.00 |  
| 格力*空调 | 大洋百货 | 4388.00 |  
| 格力*空调 | 群光广场 | 4588.00 |  
| 海尔*冰箱 | 汉商购物中心 | 3888.00 |  
+-----+-----+-----+  
6 rows in set (0.000 sec)
```

图 2.26 使用取模对个位数进行过滤

- (8) 假设品牌名不可能包含“*”号，查询所有商品的品牌。

```
-----  
select 商品名称, substr(商品名称, 1, instr(商品名称, '*') - 1) as 品牌名  
from SALE  
-----  
+-----+-----+  
| 商品名称 | 品牌名 |  
+-----+-----+  
| LEE*牛仔裤 | LEE |  
| LEE*牛仔裤 | LEE |  
| 七匹狼*夹克 | 七匹狼 |  
| 创维*电视 | 创维 |  
| 哈森*皮靴 | 哈森 |  
| 奥利奥*饼干 | 奥利奥 |  
| 奥利奥*饼干 | 奥利奥 |  
| 小天鹅*洗衣机 | 小天鹅 |  
| 得力*笔记本 | 得力 |  
| 得力*笔记本 | 得力 |  
| 晨光*中性笔 | 晨光 |  
| 晨光*中性笔 | 晨光 |  
| 格力*空调 | 格力 |  
| 格力*空调 | 格力 |  
| 海尔*冰箱 | 海尔 |  
| 老人头*t恤 | 老人头 |  
| 老干妈*辣酱 | 老干妈 |  
+-----+-----+  
17 rows in set (0.000 sec)
```

图 2.27 字符串操作

查询语句与结果如图 2.27 所示，使用 instr 和 substr 对于字符串进行剪切。

2.2.5 事务特性（选做）

T1 事务的语句如下：

```
start transaction  
update R set B=22 where A='C1';  
insert into R values ('C4', 0);  
update R set B=38 where A='C1';  
commit;
```

T2 事务的语句如下：

```
start transaction  
set transaction isolation level read committed;  
select sum(B) from R;  
select AVG(B) from R;  
commit;
```

为了同时执行两个事务，同时启动 2 个 mysql shell 进行测试。在 SQL 中事务分为 4 个等级：Read Uncommitted, Repeatable Read, Read Committed 以及 serializable，其隔离程度依次升高。在 T2 设置为 serializable 的情况下，事务从开始到结束读到的一定是同一个数据，因此无论 T1 如何改变 R，T2 计算出来的 SUM 以及 AVG 是一致的，均从 C2 开始执行的时候开始计算。如果 T2 读到的是最原始的数据，则 SUM 的结果为 150，AVG 的结果为 50，而这时与两个事务中每条语句的执行顺序无关。

2.3 任务总结

在这次实验中主要遇到的问题包括对于字符串函数的不熟悉，以及由于自身的疏漏造成的错误。在执行与字符串相关的查询任务时我使用了 MySQL 文档¹查看可以使用的函数。而在排查由于自己的疏漏而造成的偏差时（如 2.2.3 第 10 题中未使用 distinct 而造成的结果错误）则仔细的观察自己编写的 SQL 语句，人工推测其执行过程然后做出相应的修改。

最后与事务隔离级有关的实验中，不仅需要查阅文档以熟悉各个隔离级别的特性、在此隔离级下数据库的行为，还需要通过亲身实践来观察数以及数据库的真实行为，从而理解不同隔离级别之间的差异以及相同点。

¹ <https://dev.mysql.com/doc/>

3 安全性及完整性实验

3.1 任务要求

在 SQL 实验的基础上，进一步的使用 DBMS 进行权限控制以及完整性控制的实验。在此过程中掌握数据库的权限控制与完整性控制的手段，并对于存储有所了解。

3.2 完成过程

3.2.1 权限控制

(1) 在以上基础上完成教材 P155 习题 7，实现对不同用户的权限授予与回收。

a. 首先创建 2 张基本表部门和职工，使用以下的 SQL 语句：

```
create table if not exists 部门 (
    部门号 nvarchar(20) not null primary key,
    名称 nvarchar(20) not null,
    经理名 nvarchar(10),
    地址 nvarchar(50),
    电话号 nvarchar(30)
);
create table if not exists 职工 (
    职工号 nvarchar(20) not null primary key,
    姓名 nvarchar(10) not null,
    年龄 integer not null,
    职务 nvarchar(20),
    工资 decimal(8,2) default 0.0,
    部门号 nvarchar(20),
    foreign key (部门号) references 部门(部门号)
);
```

然后，使用如下的语句创建题目中涉及到的用户：

```
create user if not exists 王明@'localhost';
create user if not exists 李勇@'localhost';
create user if not exists 刘星@'localhost';
create user if not exists 张新@'localhost';
create user if not exists 周平@'localhost';
create user if not exists 杨兰@'localhost';
```

b. 用户王明对两个表有 SELECT 权限

使用以下 grantSQL 语句对于王明进行权限授予：

```
grant select on db_lab2.职工 to 王明@'localhost';
grant select on db_lab2.部门 to 王明@'localhost';
```

c. 用户李勇对两个表都有 INSERT 和 DELETE 权限

与上述语句类似，使用如下的语句对于李勇进行权限授予：

```
grant insert, delete on 职工 to 李勇@'localhost';
grant insert, delete on 部门 to 李勇@'localhost';
```

- d. 每个职工对于自己的记录有 SELECT 权限

由于每个职工仅仅对于自己的记录有 SELECT 权限，因此最好创建一个 VIEW 并对于每个用户进行单独的权限授予：

```
create view if not exists 自己 as (
    select *
    from 职工
    where 姓名=substr(user(), 1, instr(user(), '@') - 1)
);

grant select on 自己 to 王明@'localhost';
grant select on 自己 to 李勇@'localhost';
grant select on 自己 to 刘星@'localhost';
grant select on 自己 to 张新@'localhost';
grant select on 自己 to 周平@'localhost';
grant select on 自己 to 杨兰@'localhost';
```

- e. 用户刘星对职工表有 SELECT 权限，对工资字段有更新权限

此处的 grant 语句限定了能够改变的列：

```
grant select, update(工资) on 职工 to 刘星@'localhost';
```

- f. 用户张新具有修改这两个表的结构的权限

对于结构的改变使用 grant alter 语句即可实现，SQL 语句如下：

```
grant alter on 职工 to 张新@'localhost';
grant alter on 部门 to 张新@'localhost';
```

- g. 用户周平具有对两个表的所有权限，并可给其他用户权限

SQL 语句如下，授予给予其他用户权限是使用 with grant option 进行的。

```
grant all privileges on 职工 to 周平@'localhost' with grant
option;
```

- h. 用户杨兰具有从每个部门职工中 SELECT 最高工资，最低工资，平均工资的权限，但他不能查看每个人的工资

与 d 一样，创建 VIEW 可以实现对于计算过的结果的查看，同时屏蔽对于单个条目的查看，SQL 语句如下：

```
create view if not exists 工资 as (
    select 职工.部门号, 名称, max(工资), min(工资), avg(工资)
    from 职工 inner join 部门
    on 职工.部门号=部门.部门号
    group by 职工.部门号
);
grant select on 工资 to 杨兰@'localhost';
```

(2) 设计一些语句，验证上述权限控制操作的效果。

在此实验时，SQL prompt 被变更为用户名，这样更便于观察是哪个用户在执行指令。

a. 用户王明对两个表有 SELECT 权限

SQL 指令及其结果如图 3.1 所示，可以看出王明确实有查看两张表的权限

```
[王明] >select * from 职工; exit;
+-----+-----+-----+-----+-----+
| 职工号 | 姓名 | 年龄 | 职务 | 工资 | 部门号 |
+-----+-----+-----+-----+-----+
| 001   | 王明 | 23   | 研究员 | 2300.01 | 001
| 002   | 李勇 | 26   | 经理   | 4547.42 | 001
| 003   | 刘星 | 37   | 工程师 | 2234.51 | 002
| 004   | 张新 | 45   | 咨询师 | 4394.35 | 004
| 005   | 周平 | 25   | 产品经理 | 3388.45 | 003
| 006   | 杨兰 | 42   | 总监   | 4342.33 | 005
+-----+-----+-----+-----+-----+
[王明] >select * from 部门; exit;
+-----+-----+-----+-----+
| 部门号 | 名称 | 经理名 | 地址   | 电话号 |
+-----+-----+-----+-----+
| 001   | 管理 | 李勇   | 珞喻路 1038号 | 13159322791
| 002   | 研发 | 刘星   | 珞喻路 1039号 | 13342816459
| 003   | 销售 | 周平   | 珞喻路 1040号 | 13947814621
| 004   | 人力 | 张新   | 珞喻路 1041号 | 13966241962
| 005   | 产品 | 周平   | 珞喻路 1042号 | 13846834591
+-----+-----+-----+-----+
```

图 3.1 使用 select 观察王明的权限

b. 用户李勇对两个表都有 INSERT 和 DELETE 权限

使用的 SQL 语句如图 3.2 所示。从图中可以看出李勇的权限正确

```
[李勇] >insert into 职工 values("test", "测试", 18, "开发", 1200.0, "001"); exit;
[dblab] >select * from 职工; exit;
+-----+-----+-----+-----+-----+
| 职工号 | 姓名 | 年龄 | 职务 | 工资 | 部门号 |
+-----+-----+-----+-----+-----+
| 001   | 王明 | 23   | 研究员 | 2300.01 | 001
| 002   | 李勇 | 26   | 经理   | 4547.42 | 001
| 003   | 刘星 | 37   | 工程师 | 2234.51 | 002
| 004   | 张新 | 45   | 咨询师 | 4394.35 | 004
| 005   | 周平 | 25   | 产品经理 | 3388.45 | 003
| 006   | 杨兰 | 42   | 总监   | 4342.33 | 005
| test  | 测试 | 18   | 开发   | 1200.00 | 001
+-----+-----+-----+-----+
[李勇] >delete from 职工; exit;
[dblab] >select * from 职工; exit;
```

图 3.2 使用 insert 以及 delete 观察李勇的权限

c. 每个职工对于自己的记录有 SELECT 权限

```
[王明] >select * from 自己; exit;
+-----+-----+-----+-----+-----+
| 职工号 | 姓名 | 年龄 | 职务 | 工资 | 部门号 |
+-----+-----+-----+-----+-----+
| 001   | 王明 | 23   | 研究员 | 2300.01 | 001
+-----+-----+-----+-----+
[李勇] >select * from 自己; exit;
+-----+-----+-----+-----+-----+
| 职工号 | 姓名 | 年龄 | 职务 | 工资 | 部门号 |
+-----+-----+-----+-----+-----+
| 002   | 李勇 | 26   | 经理   | 4547.42 | 001
+-----+-----+-----+-----+
[刘星] >select * from 自己; exit;
+-----+-----+-----+-----+-----+
| 职工号 | 姓名 | 年龄 | 职务 | 工资 | 部门号 |
+-----+-----+-----+-----+-----+
| 003   | 刘星 | 37   | 工程师 | 2234.51 | 002
+-----+-----+-----+-----+
[张新] >select * from 自己; exit;
+-----+-----+-----+-----+-----+
| 职工号 | 姓名 | 年龄 | 职务 | 工资 | 部门号 |
+-----+-----+-----+-----+-----+
| 004   | 张新 | 45   | 咨询师 | 4394.35 | 004
+-----+-----+-----+-----+
```

图 3.3 对于 VIEW 进行选择

使用四个用户对于“自己”view 进行选择，SQL 语句以及结果如图 3.3 所示。可以看出，每个人对于自己的记录有 SELECT 权限。

d. 用户刘星对自职工表有 SELECT 权限，对工资字段有更新权限

测试的语句以及如图 3.4，与前几个测试一样，刘星具有被赋予的正确权限。

```
[刘星] >select * from 职工; exit;
+-----+
| 职工号 | 姓名 | 年龄 | 职务 | 工资 | 部门号 |
+-----+
| 001 | 王明 | 23 | 研究员 | 2300.01 | 001 |
| 002 | 李勇 | 26 | 经理 | 4547.42 | 001 |
| 003 | 刘星 | 37 | 工程师 | 2234.51 | 002 |
| 004 | 张新 | 45 | 咨询师 | 4394.35 | 004 |
| 005 | 周平 | 25 | 产品经理 | 3388.45 | 003 |
| 006 | 杨兰 | 42 | 总监 | 4342.33 | 005 |
+-----+
[刘星] >update 职工 set 工资=1234 where 职工号="001"; exit;
[刘星] >select * from 职工; exit;
+-----+
| 职工号 | 姓名 | 年龄 | 职务 | 工资 | 部门号 |
+-----+
| 001 | 王明 | 23 | 研究员 | 1234.00 | 001 |
| 002 | 李勇 | 26 | 经理 | 4547.42 | 001 |
| 003 | 刘星 | 37 | 工程师 | 2234.51 | 002 |
| 004 | 张新 | 45 | 咨询师 | 4394.35 | 004 |
| 005 | 周平 | 25 | 产品经理 | 3388.45 | 003 |
| 006 | 杨兰 | 42 | 总监 | 4342.33 | 005 |
+-----+
```

图 3.4 使用 select 和 update 对刘星的权限进行测试

e. 用户张新具有修改这两个表的结构的权限

测试的 SQL 语句及结果如图 3.5 所示。需要注意的是要将更新后的列还原以免影响后续实验。

```
[张新] >alter table 职工 add column test integer; exit;
[dblab] >select * from 职工; exit;
+-----+
| 职工号 | 姓名 | 年龄 | 职务 | 工资 | 部门号 | test |
+-----+
| 001 | 王明 | 23 | 研究员 | 1234.00 | 001 | NULL |
| 002 | 李勇 | 26 | 经理 | 4547.42 | 001 | NULL |
| 003 | 刘星 | 37 | 工程师 | 2234.51 | 002 | NULL |
| 004 | 张新 | 45 | 咨询师 | 4394.35 | 004 | NULL |
| 005 | 周平 | 25 | 产品经理 | 3388.45 | 003 | NULL |
| 006 | 杨兰 | 42 | 总监 | 4342.33 | 005 | NULL |
+-----+
[张新] >alter table 职工 drop column test; exit;
[张新] >alter table 部门 add column test integer; exit;
[dblab] >select * from 部门; exit;
+-----+
| 部门号 | 名称 | 经理名 | 地址 | 电话号 | test |
+-----+
| 001 | 管理 | 李勇 | 瑞喻路1038号 | 13159322791 | NULL |
| 002 | 研发 | 刘星 | 瑞喻路1039号 | 13342816459 | NULL |
| 003 | 销售 | 周平 | 瑞喻路1040号 | 13947814621 | NULL |
| 004 | 人力 | 张新 | 瑞喻路1041号 | 13966241962 | NULL |
| 005 | 产品 | 周平 | 瑞喻路1042号 | 13846834591 | NULL |
+-----+
[张新] >alter table 部门 drop column test; exit;
```

图 3.5 使用 alter 对张新的权限进行测试

f. 用户周平具有对两个表的所有权限，并可给其他用户权限

```
[周平] >grant all privileges on db_lab2.职工 to 王明@"localhost"; exit;
[王明] >update 职工 set 工资=4567 where 职工号="001"; exit;
[周平] >select * from 职工; exit;
+-----+
| 职工号 | 姓名 | 年龄 | 职务 | 工资 | 部门号 |
+-----+
| 001 | 王明 | 23 | 研究员 | 4567.00 | 001 |
| 002 | 李勇 | 26 | 经理 | 4547.42 | 001 |
| 003 | 刘星 | 37 | 工程师 | 2234.51 | 002 |
| 004 | 张新 | 45 | 咨询师 | 4394.35 | 004 |
| 005 | 周平 | 25 | 产品经理 | 3388.45 | 003 |
| 006 | 杨兰 | 42 | 总监 | 4342.33 | 005 |
+-----+
[周平] >revoke all privileges on db_lab2.职工 from 王明@"localhost"; exit;
```

图 3.6 让周平赋予王明权限

SQL 语句及结果如图 3.6 所示，首先让周平赋予王明一个他本没有的权限，可以看出，赋予权限后王明有了新的权限。注意完成测试后撤销了新赋予的权限。

g. 用户杨兰具有从每个部门职工中 SELECT 最高工资，最低工资，平均工资的权限，但他不能查看每个人的工资。

测试用 SQL 语句及结果如图 3.7 所示，可以看到，杨兰对于工资这一 view 有选择的权限，可以看到最高、最低、平均工资，但是不能选择单独的工资。

```
[杨兰] >select * from 工资; exit;
+-----+-----+-----+-----+
| 部门号 | 名称   | max(工资) | min(工资) | avg(工资) |
+-----+-----+-----+-----+
| 001    | 管理   | 4567.00  | 4547.42  | 4557.210000 |
| 002    | 研发   | 2234.51  | 2234.51  | 2234.510000 |
| 003    | 销售   | 3388.45  | 3388.45  | 3388.450000 |
| 004    | 人力   | 4394.35  | 4394.35  | 4394.350000 |
| 005    | 产品   | 4342.33  | 4342.33  | 4342.330000 |
+-----+-----+-----+-----+
[杨兰] >select 工资 from 职工; exit;
ERROR 1142 (42000): SELECT command denied to user '杨兰'@'localhost' for table '职工'
```

图 3.7 使用 select 对于杨兰的权限进行测试

3.2.2 完整性控制

(1) 以教材 P173 习题 6 中的表为基础，定义以下完整性约束：

a. 定义每个模式的主码。

使用 alter 语句定义每个模式的主码：

```
alter table 职工 add primary key(职工号);
alter table 部门 add primary key(部门号);
```

b. 定义参照完整性约束，要求：当部门号改变时，该部门职工记录的部门号也做出相应改变；当部门被删除时，将该部门职工记录的部门号置空。

使用 cascade 和 set null 约束的 foreign 可以来完成这一约束：

```
alter table 职工 add foreign key(部门号) references 部门(部
门号)
on update cascade
on delete set null;
```

c. 职工年龄不能超过 60 岁。

使用 constraint check 完成这一限制：

```
alter table 职工 add constraint check(年龄<=60);
```

d. 职工姓名和部门名称都不允许取空值。

重新定义两列，加上 not null 的限制即可：

```
alter table 职工 modify 姓名 nvarchar(10) not null;
alter table 部门 modify 名称 nvarchar(20) not null;
```

e. 部门名称不允许重复。

同样，使用 constraint 来完成唯一性约束：

```
alter table 部门 add constraint unique(名称);
```

f. 同一个部门不应该有一个以上的经理，并且从职工表查询得到的部门经理的

信息应该与从部门表查询得到的信息一致。如果用户的某个操作可能导致数据不一致，则以职工表的信息为准自动修正。

改约束十分复杂，无法通过简单的约束语句实现，只能桶盖触发器实现。为了避免无限循环触发，使用了 recurseLock 变量来保证这一点：

```
delimiter //
create trigger if not exists 职工插入触发器
before insert
on 职工 for each row
begin
    if(@recurseLock is null) then
        set @recurseLock=1;
        if (new.职务='经理' and new.部门号 is not null) then
            if (exists (select 部门号
                        from 职工
                        where 部门号=new.部门号 and 职务='经理'))
then
                signal sqlstate '45000'
                set message_text='此部门已有经理';
            else
                update 部门 set 经理名=new.姓名 where 部门号
=new.部门号;
            end if;
        end if;
        set @recurseLock=null;
    end if;
end;//

create trigger if not exists 职工删除触发器
after delete
on 职工 for each row
begin
    if(@recurseLock is null) then
        set @recurseLock=1;
        update 部门
        set 经理名=null
        where 经理名=old.姓名 and 部门号=old.部门号;
        set @recurseLock=null;
    end if;
end;//

create trigger if not exists 职工更新触发器
after update
on 职工 for each row
begin
    if(@recurseLock is null) then
```

```

set @re recurseLock=1;
if (old.职务 is null or new.职务 is null or
    old.部门号 is null or new.部门号 is null or
    old.职务!=new.职务 or old.部门号!=new.部门号)
then
    -- a delete
    update 部门
    set 经理名=null
    where 经理名=old.姓名 and 部门号=old.部门号;
    -- an insert
    -- insert conflit only occurs when 职务 or
    部门 updates
        if (new.职务='经理' and new.部门号 is not null)
    then
        if ((select count(*)
              from 职工
              where 部门号=new.部门号 and 职务='经理'
            ) > 1) then
            signal sqlstate '45000'
            set message_text='此部门已有经理';
        else
            update 部门 set 经理名=new.姓名 where 部
            门号=new.部门号;
            end if;
        end if;
    end if;
    set @re recurseLock=null;
end if;
end;//



create trigger if not exists 部门插入触发器
before insert
on 部门 for each row
begin
    if(@re recurseLock is null) then
        set @re recurseLock=1;
        if (new.经理名 is not null) then
            signal sqlstate '45000'
            set message_text='经理名称必须为空, 请在职工表中更
            新部门经理信息';
        end if;
        set @re recurseLock=null;
    end if;
end;//



create trigger if not exists 部门更新触发器

```

```

after update
on 部门 for each row
begin
    if(@recurseLock is null) then
        set @recurseLock=1;
        if (old.经理名 is null or new.经理名 is null or
old.经理名!=new.经理名) then
            signal sqlstate '45000'
            set message_text='经理名一旦修改即会与职工表冲突,
请修改职工表';
        end if;
        set @recurseLock=null;
    end if;
end;//

delimiter ;

```

(2) 构造一些 SQL 语言，验证以上完整性控制机制能否正确实施。

a. 定义参照完整性约束，要求：当部门号改变时，该部门职工记录的部门号也做出相应改变；当部门被删除时，将该部门职工记录的部门号置空。

使用 update 语句更新部门表，然后查看职工表中的内容，如图 3.8 所示。可以看到职工表中的项目也被递归更新了，说明参照完整性正确。

职工号	姓名	年龄	职务	工资	部门号
001	蔡荣	20	经理	5483.54	001
002	廖宁	35	员工	6599.65	001
003	孟羚	39	员工	5886.58	001
004	高恒	24	经理	5784.57	002
005	王光	44	员工	5176.51	002
006	廖炎	55	员工	6326.63	002
007	谢承	22	经理	6113.61	003
008	姚砾	43	员工	6481.64	003
009	卢旭	39	员工	5446.54	003
010	余棠	28	经理	5320.53	999
012	白孜	19	员工	6834.68	999
018	常汉	48	员工	5703.57	999

图 3.8 检验参照完整性约束

b. 职工年龄不能超过 60 岁。

使用 update 语句尝试将一个职工的年龄修改至 60 岁，可以看到违反了限制不能修改，如图 3.9 所示。

```
[dblab] >update 职工 set 年龄=90 where 职工号="001"; exit;
ERROR 4025 (23000): CONSTRAINT `CONSTRAINT_1` failed for `db_lab2`.`职工`
```

图 3.9 尝试修改职工年龄到 60 岁以上

c. 职工姓名和部门名称都不允许取空值。

测试的 SQL 语句如图 3.10 所示。可以看待，无论名称是空值还是部门名称是空值都不能正确插入。

```
[dblalb] >insert into 职工 values("999", null, 12, "test", 123.00, "001");
ERROR 1048 (23000): Column '姓名' cannot be null
[dblalb] >insert into 部门 values("999", null, null, "123456789"); exit;
ERROR 1048 (23000): Column '名称' cannot be null
```

图 3.10 尝试插入带空值的项

d. 部门名称不允许重复。

尝试插入一个与已有部门同名的部门，如图 3.11 所示，可以看到由于之前设置的 constraint 而无法插入。

```
[dblalb] >insert into 部门 values("999", "管理", null, "123456789"); exit;
ERROR 1062 (23000): Duplicate entry '管理' for key '名称'
```

图 3.11 尝试插入重复的部门

e. 同一个部门不应该有一个以上的经理，并且从职工表查询得到的部门经理的信息应该与从部门表查询得到的信息一致。如果用户的某个操作可能导致数据不一致，则以职工表的信息为准自动修正。

图 3.12 ~ 图 3.15 分别对于触发器的一系列测试，此外还有更完整的对于编写的触发器的测试由于篇幅原因未能给出。由于题目“以职工表信息为准自动修正”是模糊的，它未能清晰规定在各种情况下限制，因此在此实验中采取了一种较为保守的方法，禁止由触发器修改员工表，同时加上递归锁，从而避免可能的无限连锁触发，同时又保证了所有的约束均满足且数据库不会因为触发器带来的限制而造成功能上的损失。

```
[dblalb] >insert into 职工 values("999", "test", 12, "经理", 123.00, "001"); exit;
ERROR 1644 (45000): 此部门已有经理
[dblalb] >update 职工 set 职务="经理" where 职工号="002"; exit;
ERROR 1644 (45000): 此部门已有经理
```

图 3.12 尝试产生一个以上的经理

```
[dblalb] >insert into 部门 values("100", "测试部", "白孜", "123456789"); exit;
ERROR 1644 (45000): 经理名称必须为空，请在职工表中更新部门经理信息
[dblalb] >insert into 部门 values("100", "测试部", null, "123456789"); exit;
```

图 3.13 尝试在部门中插入经理

```
[dblalb] >update 部门 set 经理名="白孜" where 部门号="100"; exit;
ERROR 1644 (45000): 经理名一旦修改即会与职工表冲突，请修改职工表
```

图 3.14 尝试修改部门经理名称

```
[dblalb] >insert into 职工 values("100", "测试1", 30, "员工", 5000.00, "100"); exit;
[dblalb] >insert into 职工 values("101", "测试2", 30, "经理", 6000.00, "100"); exit;
```

图 3.15 尝试在职工表中插入职工和经理

3.2.3 存储过程（选做）

(1) 根据输入的部门名称返回一个部门的平均工资；

对应的存储过程 SQL 语句如下：

```

create procedure if not exists 获得平均工资 (
    in 部门名称 nvarchar(20),
    out 平均工资 decimal(8, 2)
)
select avg(工资) into 平均工资
from 职工 inner join 部门
on 职工.部门号=部门.部门号
where 名称=部门名称;

```

在每一次调用此过程时调用 SELECT 函数返回对应部门的平均工资，测试用 SQL 语句以及返回结果如图 3.16 所示，其中最后一个 SQL 语句用于作为正确性比对。可以看到，在一定的精度范围内，此存储过程功能正确。

```

[dblab] >call 获得平均工资("管理", @管理_平均工资); select @管理_平均工资; exit;
+-----+
| @管理_平均工资 |
+-----+
| 5989.92 |
+-----+
[dblab] >call 获得平均工资("IT", @IT_平均工资); select @IT_平均工资; exit;
+-----+
| @IT_平均工资 |
+-----+
| 5762.57 |
+-----+
[dblab] >call 获得平均工资("人力", @人力_平均工资); select @人力_平均工资; exit;
+-----+
| @人力_平均工资 |
+-----+
| 6013.93 |
+-----+
[dblab] >call 获得平均工资("销售", @销售_平均工资); select @销售_平均工资; exit;
+-----+
| @销售_平均工资 |
+-----+
| 5952.93 |
+-----+
[dblab] >select 部门号, avg(工资) from 职工 group by 部门号; exit;
+-----+
| 部门号 | avg(工资) |
+-----+
| 001 | 5989.923333 |
| 002 | 5762.570000 |
| 003 | 6013.930000 |
| 004 | 5952.926667 |
+-----+

```

图 3.16 平均工资存储过程调用测试

(2) 对所有员工进行工资调整：IT 部门的员工工资上调 50%；销售部门的员工工资上调 20%；其他部门的员工工资上调 10%；部门经理的工资上调 50%；同时满足两个调薪条件的员工按调薪幅度高者调薪。

加薪的存储过程 SQL 如下：

```

create procedure if not exists 加薪 ()
update 职工 inner join 部门
on 职工.部门号=部门.部门号
set 工资=工资*case
    when 职务='经理' then 1.5
    when 部门.名称='IT' then 1.5

```

```

when 部门.名称='销售' then 1.2
else 1.1
end;

```

在每一次调用时调用 update 对于每一个部门进行加薪，并通过 case 语句结合 when 的顺序达到优先级的效果。测试用 SQL 语句以及执行结果如图 3.17 所示，图中调用前后各查看了一次职工表，可以看到，加薪存储过程功能正确。

[dblab] >select * from 职工; exit;					
职工号	姓名	年龄	职务	工资	部门号
001	蔡荣	20	经理	5483.54	001
002	廖宁	35	员工	6599.65	001
003	孟羚	39	员工	5886.58	001
004	高恒	24	经理	5784.57	002
005	王光	44	员工	5176.51	002
006	廖炎	55	员工	6326.63	002
007	谢承	22	经理	6113.61	003
008	姚砾	43	员工	6481.64	003
009	卢旭	39	员工	5446.54	003
010	余棠	28	经理	5320.53	004
012	白孜	19	员工	6834.68	004
018	常汉	48	员工	5703.57	004

[dblab] >call 加薪; exit; [dblab] >select * from 职工; exit;					
职工号	姓名	年龄	职务	工资	部门号
001	蔡荣	20	经理	8225.31	001
002	廖宁	35	员工	7259.62	001
003	孟羚	39	员工	6475.24	001
004	高恒	24	经理	8676.86	002
005	王光	44	员工	7764.77	002
006	廖炎	55	员工	9489.95	002
007	谢承	22	经理	9170.42	003
008	姚砾	43	员工	7129.80	003
009	卢旭	39	员工	5991.19	003
010	余棠	28	经理	7980.80	004
012	白孜	19	员工	8201.62	004
018	常汉	48	员工	6844.28	004

图 3.17 加薪存储过程调用测试

3.3 任务总结

在进行这一实验时，主要遇到了以下问题：

首先是创建的用户无法创建新用户的权限问题，即使使用了 grant all 也是如此，经过查阅文档发现创建用户需要有 mysql.user 这张表的权限。使用 root 用户赋予了正确的权限后可以创建用户了。

其二是创建存储过程失败，经查阅文档发现使用 procedure 中有多条语句时需要修改 delimiter 才能正确创建。经过一番努力将多个查询语句合成一条之后也解决了这一问题。

4 综合实践任务

4.1 系统设计目标

随着学科与时代的发展，大学课程种类增多，课程以及有关的作业、任务等的种类也越来越多，设计并实现一个有条理的课程管理系统显得较为重要。这一实验的目标则是设计一个功能完善，性能优良的课程、选课、授课、成绩等信息的管理系统。

此系统需要提供师生的选课、授课、布置任务，成绩管理等多方面的需求，同时达到较高的性能，又不失安全性。此外，还需要进行模块化的设计，使得这一系统随着时间的推移可以自由的增删需求而无需过多的更改，同时与第三方程序保持一定的兼容性。

4.2 需求分析

采用 B/S 或 C/S 模式实现一个成绩管理系统。完成课程、学生、教师、选课、授课、成绩等信息的管理。功能上需要提供管理员、学生、教师三类用户的录入、查询界面，课程可以有多次作业或测试，作业或测试的数量不能由系统预先指定，而应由教师本人指定自己所负责课程的作业及测试的数量，系统应支持等级评定，允许对不同的级别指定分隔点，提供学生、课程等不同角度的成绩统计功能等等。

对于功能需求进行翻译，则在数据库的层面，需要收这些关系约束：

- (1) 学生可以选修多门课程，一门课程可以被多个学生选修
- (2) 教师可以教授多门课程，一门课程可以被多个教师教授
- (3) 一位教师可以布置多个任务，一个任务只对应一门课程
- (4) 学生可以完成多个教师布置的任务，多个学生可以完成同一个任务
- (5) 学生和教师可以提交多个申请（如选课、开课、授课申请等）
- (6) 一个申请只由一个管理员批准，一个管理员可以批准多个申请

此外，三种角色自己具有的权限如下：

- (1) 教师用户需要获得有关自身的账号信息、自己教授的课程信息、自己布置的任务的信息，并可以为完成的任务打分，可以提交教师相关的申请。
- (2) 学生用户需要获得有关自身的账号信息、可选的课程信息、已选课程有关的任务信息，并可以完成、提交相应的任务，可以提交学生相关申请。
- (3) 管理员可以获得并修改所有关系的信息，并具有批准申请的能力。

在性能方面，需要一次能够处理大量的请求，并且在高负荷情况下对于请求可以缓存处理，此外，当请求不能得到及时的处理时需要向客户端返回必要的信息。

在安全性方面，需要保证每一个角色只对于自己的个人信息有访问权限，只

对于自己这一角色提供的服务有调用权限，此外没有其他的权限，而在客户端图形界面之外不能直接连接数据库，在客户端内无法通过注入进行攻击。

4.3 总体设计

总体上而言，系统为 C/S 结构包含 5 个部分：数据库、服务端、教师客户端、学生客户端、管理员客户端。只有服务端能够直接访问数据库，而客户端只能通过与服务端的沟通来间接的访问数据库，从而达到较高的安全性。总体架构图如图 4.1 所示。

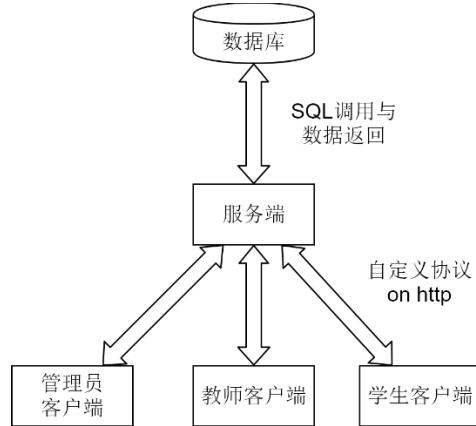


图 4.1 整体结构

图中，自定义协议是一套使用 json 包装的，在 HTTP 上进行传输的协议，使用这一套协议可以最大限度的保证数据库的安全性。

4.4 数据库设计

将功能需求转换为 E-R 图，则如图 4.2 所示。图中，所有的主码已用下划线标出。

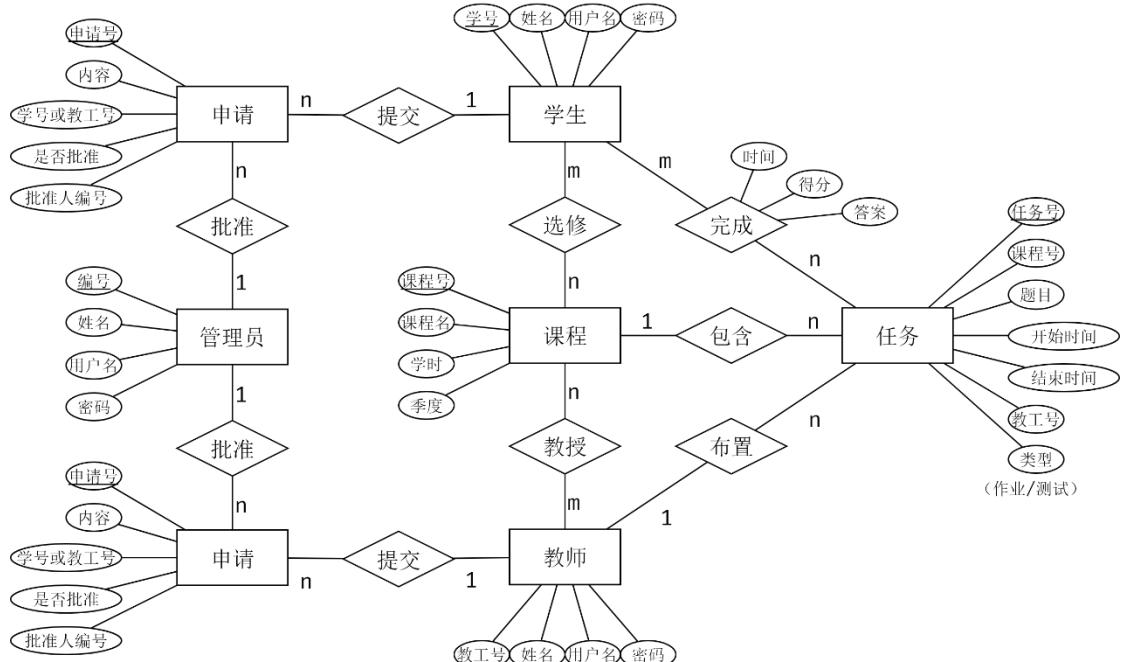


图 4.2 数据库实体-关系图

接下来，将 E-R 图转换为实际的数据库结构。图中的每一个实体对应一张表，1 对 n 关系在 n 端连接的表中引用 1 端所在表的主码作为外码，n 对 n 关系则将关系本身构建为一张独立的表，这张独立的表含有两个实体的主码。以此构建的数据库中含有的表如表 4.1 所示。

表 4.1 数据库中所含有的表及其说明

表名	说明
Course	课程列表
Teacher	教师列表
Student	学生列表
Manager	管理员列表
Task	教师布置的任务列表
StudentCourse	学生选课表
TeacherCourse	教师授课表
StudentTask	学生完成任务列表
StudentApplication	学生申请列表
TeacherApplication	教师申请列表

为了能够更为完整的展示各表的内容以及表之间而约束关系，列出各表的创建语句如下：

课程列表 Course 的各字段以及说明如表 4.2:

表 4.2 Course 各字段说明

字段	类型	说明
id	char(10)	课程号
name	varchar(50)	课程名
span	int(11)	时长
season	int(11)	开课季度

创建列表的 SQL 语句为：

```
CREATE TABLE `Course` (
  `id` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `name` varchar(50) CHARACTER SET utf8 NOT NULL,
  `span` int(11) DEFAULT NULL,
  `season` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

管理员列表 Manager 的各字段以及说明如表 4.3:

表 4.3 Manager 各字段说明

字段	类型	说明

id	char(10)	管理员编号
name	varchar(50)	管理员名称
username	varchar(20)	用户名
password	char(64)	密码 (hash, 下同)

创建列表的 SQL 语句为:

```
CREATE TABLE `Manager` (
  `id` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `name` varchar(50) CHARACTER SET utf8 NOT NULL,
  `username` varchar(20) COLLATE utf8mb4_unicode_ci
  DEFAULT NULL,
  `password` char(64) COLLATE utf8mb4_unicode_ci NOT
  NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

学生列表 Student 的各字段以及说明如表 4.4:

表 4.4 Student 各字段说明

字段	类型	说明
id	char(10)	学号
name	varchar(50)	姓名
username	varchar(20)	用户名
password	char(64)	密码

创建列表的 SQL 语句为:

```
CREATE TABLE `Student` (
  `id` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `name` varchar(50) CHARACTER SET utf8 NOT NULL,
  `username` varchar(20) COLLATE utf8mb4_unicode_ci
  DEFAULT NULL,
  `password` char(64) COLLATE utf8mb4_unicode_ci NOT
  NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

教师列表 Teacher 的各字段以及说明如表 4.5:

表 4.5 Teacher 各字段说明

字段	类型	说明
id	char(10)	教工号
name	varchar(50)	姓名
username	varchar(20)	用户名

password	char(64)	密码
----------	----------	----

创建列表的 SQL 语句为:

```
CREATE TABLE `Teacher` (
  `id` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `name` varchar(50) CHARACTER SET utf8 NOT NULL,
  `username` varchar(20) COLLATE utf8mb4_unicode_ci
  DEFAULT NULL,
  `password` char(64) COLLATE utf8mb4_unicode_ci NOT
NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

任务列表 Task 的各字段以及说明如表 4.6:

表 4.6 Task 各字段说明

字段	类型	说明
id	char(10)	任务编号
courseId	char(10)	所属课程 Id
teacherId	char(10)	布置此任务的教师编号
question	text	任务内容
start	timestamp	开始时间
end	timestamp	截止时间
type	char(10)	任务类型

创建列表的 SQL 语句为:

```
CREATE TABLE `Task` (
  `id` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `courseId` char(10) COLLATE utf8mb4_unicode_ci DEFAULT
NULL,
  `teacherId` char(10) COLLATE utf8mb4_unicode_ci DEFAULT
NULL,
  `question` text COLLATE utf8mb4_unicode_ci,
  `start` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
  `end` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  `type` char(10) COLLATE utf8mb4_unicode_ci DEFAULT
NULL,
  PRIMARY KEY (`id`),
  KEY `teacherId`(`teacherId`),
  KEY `courseId`(`courseId`, `teacherId`),
  CONSTRAINT `Task_ibfk_1` FOREIGN KEY (`courseId`, `teacherId`)
    REFERENCES `TeacherCourse`(`courseId`, `teacherId`)
    ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

```
COLLATE=utf8mb4_unicode_ci;
```

学生选课列表各字段及说明如表 4.7:

表 4.7 StudentCourse 各字段说明

字段	类型	说明
studentId	char(10)	学生 Id
courseId	char(10)	课程 Id

创建列表的 SQL 语句为:

```
CREATE TABLE `StudentCourse` (
  `studentId` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `courseId` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  PRIMARY KEY (`courseId`, `studentId`),
  KEY `studentId` (`studentId`),
  CONSTRAINT `StudentCourse_ibfk_1` FOREIGN KEY
(`studentId`) REFERENCES `Student` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE,
  CONSTRAINT `StudentCourse_ibfk_2` FOREIGN KEY
(`courseId`) REFERENCES `Course` (`id`) ON DELETE CASCADE
ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

学生完成任务列表各字段及说明如表 4.8:

表 4.8 StdudentTask 各字段说明

字段	类型	说明
studentId	char(10)	学生 Id
taskId	char(10)	任务 Id
answer	text	答案
score	int(11)	分数
finsihTime	timestemp	完成时间

创建列表的 SQL 语句为:

```
CREATE TABLE `StudentTask` (
  `studentId` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `taskId` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `answer` text COLLATE utf8mb4_unicode_ci,
  `score` int(11) DEFAULT NULL,
  `finishTime` timestamp NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`studentId`, `taskId`),
  KEY `taskId` (`taskId`),
```

```

CONSTRAINT `StudentTask_ibfk_1` FOREIGN KEY
(`studentId`) REFERENCES `Student` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE,
CONSTRAINT `StudentTask_ibfk_2` FOREIGN KEY (`taskId`)
REFERENCES `Task` (`id`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;

```

学生申请列表 StudentApplication 的各字段以及说明如表 4.9:

表 4.9 StudentApplication 各字段说明

字段	类型	说明
id	char(10)	申请编号
statement	text	申请内容
studentId	char(10)	申请人学号
approved	bool	是否同意
managerId	char(10)	处理人编号

创建列表的 SQL 语句为:

```

CREATE TABLE `StudentApplication` (
`id` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
`statement` text COLLATE utf8mb4_unicode_ci,
`studentId` char(10) COLLATE utf8mb4_unicode_ci NOT
NULL,
`approved` tinyint(1) DEFAULT NULL,
`managerId` char(10) COLLATE utf8mb4_unicode_ci DEFAULT
NULL,
PRIMARY KEY (`id`),
KEY `studentId` (`studentId`),
KEY `managerId` (`managerId`),
CONSTRAINT `StudentApplication_ibfk_1` FOREIGN KEY
(`studentId`) REFERENCES `Student` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE,
CONSTRAINT `StudentApplication_ibfk_2` FOREIGN KEY
(`managerId`) REFERENCES `Manager` (`id`) ON DELETE SET
NULL ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;

```

教师申请列表 TeacherApplication 的各字段以及说明如表 4.10:

表 4.10 TeacherApplication 各字段说明

字段	类型	说明
id	char(10)	申请编号
statement	text	申请内容
teacherId	char(10)	申请人教工号

approved	bool	是否同意
managerId	char(10)	处理人编号

创建列表的 SQL 语句为：

```
CREATE TABLE `TeacherApplication` (
  `id` char(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `statement` text COLLATE utf8mb4_unicode_ci,
  `teacherId` char(10) COLLATE utf8mb4_unicode_ci NOT
NULL,
  `approved` tinyint(1) DEFAULT NULL,
  `managerId` char(10) COLLATE utf8mb4_unicode_ci DEFAULT
NULL,
  PRIMARY KEY (`id`),
  KEY `teacherId` (`teacherId`),
  KEY `managerId` (`managerId`),
  CONSTRAINT `TeacherApplication_ibfk_1` FOREIGN KEY
(`teacherId`) REFERENCES `Teacher` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE,
  CONSTRAINT `TeacherApplication_ibfk_2` FOREIGN KEY
(`managerId`) REFERENCES `Manager` (`id`) ON DELETE SET
NULL ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

4.5 详细设计与实现

4.5.1 服务端实现

此系统的核心部分为服务端，其接受服务端的所有请求，分析并缓存请求，处理请求，分析并处理数据库操作结果并返回结果。对于权限的管理，请求完整的验证以及数据库操作的处理均在服务端完成。服务端的具体架构如图 4.3 所示。

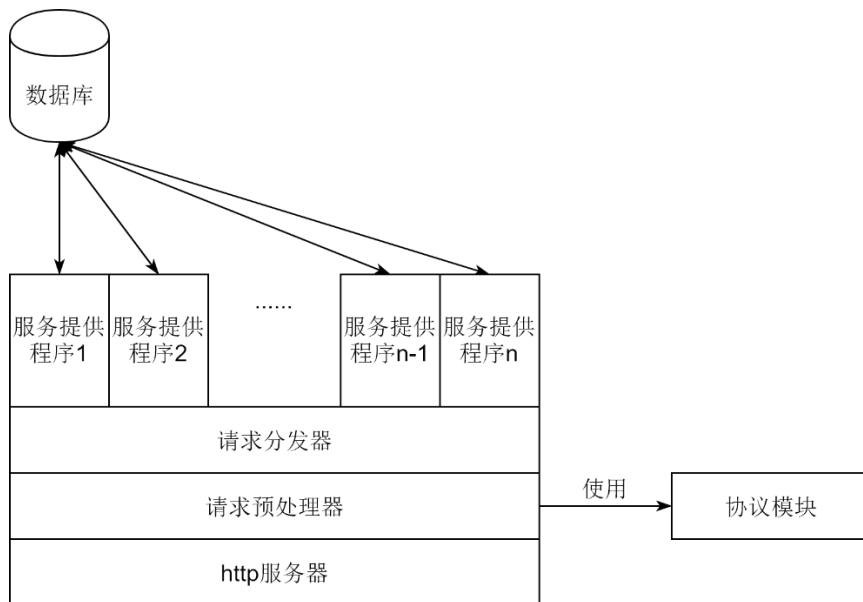


图 4.3 服务端详细结构

图中，协议模块是一个客户端与服务端共同使用的 C++ 模块，其中定义了服务端提供的服务类型、服务的请求格式、对于服务请求的一些基本操作、客户端的角色、客户端的请求内容等。在定义好服务端的 http 服务器以及客户端的 http 客户端后，双方直接通过此协议模块定义的协议层进行通信，而底层的 http 模块对于客户端则是透明的，这样，与底层分离协议便十分易于修改。

请求预处理器对于客户端发来的请求进行预处理，检验请求是否完整，以及权限是否匹配（即客户角色是否能够使用请求）。为了防止客户端伪造身份，使用客户端发送过来的密码（hash 后）进行身份验证。这样确实、加重了数据库的负担，但在更进一步的修改中可以只在第一次登录时进行验证，此后服务端向客户端发送随机动态口令并使用加密过的动态口令进行身份验证，这样可以极大地减少数据库的负担。但由于时间的限制，在本次的实现中未采用这种较为复杂的方法。

经过预处理器的初步验证后，请求分发器将请求发送给（可能的）存在于多个线程上的锁个服务提供程序，以达到并行处理的效果。但在实际的测试中，CPU 的处理速度并不是瓶颈，对于数据库的访问才是，因此使用多线程对于处理速度的提升不大，而对于数据库的存储优化则会产生比较大的影响。

最后模块化的服务提供程序初步进行数据库访问，处理数据库的返回结果后及结果返回给客户端。在协议处理时，为了避免注入攻击采用参数化查询，使用预编译的 sql 指令进行数据库的操作。

服务端提供的程序如表 所示。

服务	说明	SQL 语句
login	登录	<code>select * from Student/Teacher/Manager where id=:realId</code>
getCourse	获得课程列表	<code>select * from Course</code>
getRegisteredCourse	学生获得已选课程	<code>select * from StudentCourse inner join Course on StudnetCourse.courseId = Course.id where studentId=:realId</code>
getRelaventTaskList	获得与已选课程有关的任务列表	<code>select * from (select Task.id, Course.name, StudnetCourse.studnetId as sid from (Task inner join Course on Task.courseId = Course.id inner join</code>

		StudnetCourse on StudnetCourse.courseId = Course.id) where StudentCourse.studnetId = :stuId) as base left join StudentTask on (base.sid = StudnetTask.studentId and base.id = StudentTask.taskId)
getSpecificTask	获得特定的任务	select * from Task where id = :taskId
submitTask	提交任务	insert into StudentTask values(:sId, :tId, :and, null, :time)
registerCourse	注册课程	insert into StudentApplication values(:aId, :stmt, :sId, null, null)
getRelaventAppList	获得已提交的申请	select * from StudentApplication/ TeacherApplicatoin where studentId/teacherId = :sId/:tId
unRegisterCourse	取消课程注册	insert into StudentApplication/ TeacherApplication values (:aId, :stmt, :sId, null, null)
getTeachingCourse	获得正在教授的课程列表	select * from TeacherCourse inner join Course where Course.id = TeacherCourse.courseId where id = :tId
getAssignedTaskList	获得自己布置的任务	select * from Task inner join Course on Task.courseId = Course.id where Task.teacherId = :id
selectCourseToTeach	注册教授课程	insert into

		TeacherApplication values (:aId, :stmt, :sId, null, null)
getStudentsForTask	获得关于某个任务的学生列表	select Student.id, Student.name, StudentTask.score from Student inner join StudentTask on Student.id = StudentTask.studentId where StudentTask.studentId = :sId
getSpecificAnswer	获得某个学生针对某个任务的答案	select * from StudentTask where (studentId, taskId) = (:sId, :tId)
gradeStudent	给某个学生的某个任务打分	update StudentTask set score = :grade where (studentId, taskId) = (:sId, :tId)
addTask	添加任务	insert into Task values (:id, :cId, :tId, :q, :start, :end, :type)
getStudents	获得学生列表	select * from Student
getTeachers	获得教师列表	select * from Teacher
approveApplication	批准申请	1. select * from StudentApplication/ TeacherApplication where id = :aId 2. 组成并执行预先存储的语句 3. update StudentApplication/ TeacherApplication set managerId = :mId, isApproved = :acc where id = :aId
declineApplication	拒绝申请	update StudentApplication/ TeacherApplication set managerId = :mId, isApproved = :acc where id = :aId
getTeacherAppList	获得教师申请列表	select * from TeacherApplication where

```

isApproved is null
getStudentAppList 获得学生申请 select * from
                  列表 StudentApplication where
                  isApproved is null

```

其中，以冒号开头的为占位符，在运行时动态绑定值并执行

4.5.2 客户端实现

客户端的架构如图 4.4 所示。其中，用户通过前端与程序进行交互，前端传递信息给后端，后端使用协议模块构造请求后通过 http 客户端发送给服务器，等待服务器响应后处理服务器返回的数据，通过协议模块进行解析后在界面上进行显示。

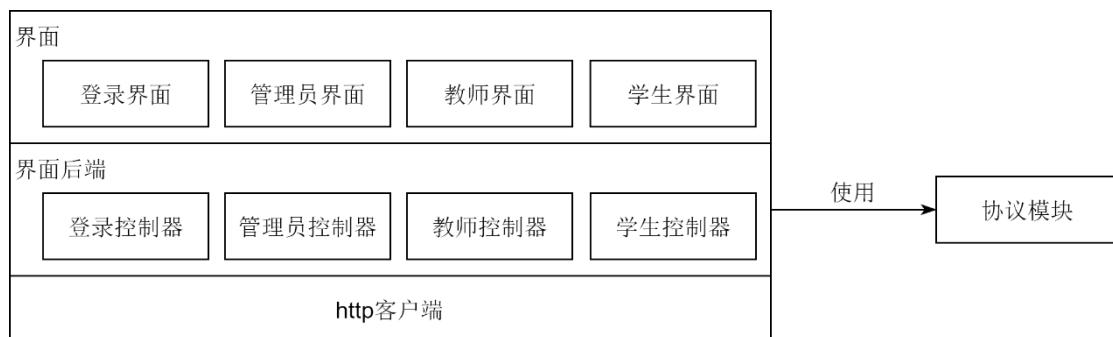


图 4.4 客户端架构

由于服务端已经定义好了所有的服务，客户端只需通过协议模块构造请求并发送即可，客户端剩下的任务就是给用户一个简单易懂的，易于交互的界面。客户端的 4 个模块的界面设计如下，其界面设计也反映了客户端给用户提供的功能。由于使用了 C/S 架构，且由于时间上的关系，服务器提供的一些服务并没有在客户端实现，不过大部分需求均已实现。

其中，登录界面如图 4.5 所示，上方输入用户名和密码，下方选择登录角色。

图 4.5 登录界面

一个不成功的登录如图 4.6 所示，在下方会提示错误原因。不止是登录界面，所有的界面中均有这样一个状态条显示当前的状态。

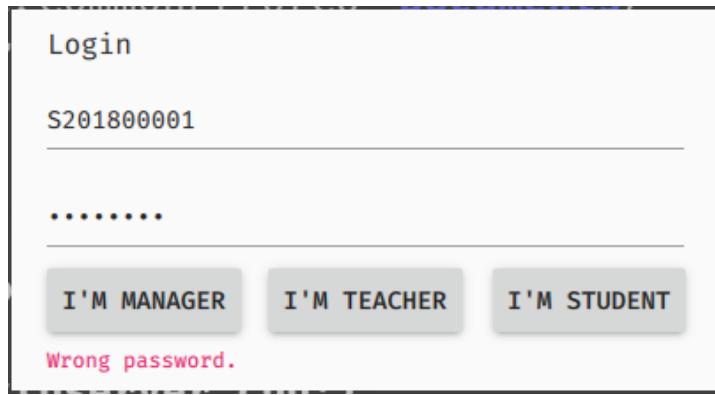


图 4.6 不成功的登录

学生登录后的界面如图 4.7 所示，主页的 4 个信息面板分别显示个人信息、已选课程、课程相关任务以及已提交的申请。

SELF	COURSE	TASK
Personal Information Name: 李德 User: S201800001 Id: S201800001	Choosen Courses C201800008: 模拟电子技术 (二) C201800010: 数字电路与逻辑设计 (一) C201800018: 汇编语言程序设计 C201800035: JAVA语言程序设计	
Tasks 模拟电子技术 (二) : W201800021, scores: no score 模拟电子技术 (二) : W201800062, scores: no score 模拟电子技术 (二) : W201800059, scores: no score 模拟电子技术 (二) : W201800174, scores: no score 数字电路与逻辑设计 (一) : W201800018, scores: no score 数字电路与逻辑设计 (一) : W201800044, scores: no score 数字电路与逻辑设计 (一) : W201800169, scores: no score	Applications	

图 4.7 学生登录后界面

点击到 Course tab，进入选课界面。选课界面上有一排过滤栏以及一个刷线按钮和一个选课按钮。在刷新按钮按下后，会向服务器发送请求将本地的课程与服务器的课程进行同步，更新本地课程到最新。通过顶部的过滤栏可以按照课程编号、名称以及时长进行过滤，如图 4.9 所示。这样对于课程的筛选而言更为方便。

SELF		COURSE			TASK	
id		name	span		REFRESH	REGISTER
select	Course Id	Name	Span	Season		
<input type="checkbox"/>	C20180001	线性代数（一）	40	5		
<input type="checkbox"/>	C20180002	概率论与数理统计...	40	1		
<input type="checkbox"/>	C20180003	复变函数与积分变换	40	5		
<input type="checkbox"/>	C20180004	人文社科类选修课程	160	2		
<input type="checkbox"/>	C20180005	C语言程序设计	48	3		
<input type="checkbox"/>	C20180006	C语言程序设计实验	32	1		
<input type="checkbox"/>	C20180007	电路理论（五）	64	4		
<input checked="" type="checkbox"/>	C20180008	模拟电子技术（二）	48	1		
<input type="checkbox"/>	C20180009	信号与线性系统	40	4		
<input checked="" type="checkbox"/>	C20180010	数字电路与逻辑设...	56	1		
<input type="checkbox"/>	C20180011	数字电路与逻辑设...	16	5		
<input type="checkbox"/>	C20180012	计算机通信与网络	40	4		
<input type="checkbox"/>	C20180013	计算机通信与网络...	32	4		
<input type="checkbox"/>	C20180014	离散数学（一）	40	6		
<input type="checkbox"/>	C20180015	离散数学（二）	40	3		
<input type="checkbox"/>	C20180016	数据结构	48	8		
<input type="checkbox"/>	C20180017	数据结构实验	32	8		
<input checked="" type="checkbox"/>	C20180018	汇编语言程序设计	24	7		
<input type="checkbox"/>	C20180019	汇编语言程序设计...	32	1		
<input type="checkbox"/>	C20180020	计算机组成原理	48	8		
<input type="checkbox"/>	C20180021	计算机组成原理实验	16	7		
<input type="checkbox"/>	C20180022	操作系统原理	56	7		
<input type="checkbox"/>	C20180023	操作系统原理实验	16	4		
<input type="checkbox"/>	C20180024	数据库系统原理	48	7		
<input type="checkbox"/>	C20180025	数据库系统原理实践	32	3		

图 4.8 刷新后的选课界面

SELF		COURSE			TASK	
id		name	span		REFRESH	REGISTER
select	Course Id	Name	Span	Season		
001	<input type="checkbox"/>	C20180001	线性代数（一）	40	5	
	<input checked="" type="checkbox"/>	C20180010	数字电路与逻辑设...	56	1	
	<input type="checkbox"/>	C20180011	数字电路与逻辑设...	16	5	
	<input type="checkbox"/>	C20180012	计算机通信与网络	40	4	
	<input type="checkbox"/>	C20180013	计算机通信与网络...	32	4	
	<input type="checkbox"/>	C20180014	离散数学（一）	40	6	
	<input type="checkbox"/>	C20180015	离散数学（二）	40	3	
	<input type="checkbox"/>	C20180016	数据结构	48	8	
	<input type="checkbox"/>	C20180017	数据结构实验	32	8	
	<input checked="" type="checkbox"/>	C20180018	汇编语言程序设计	24	7	
	<input type="checkbox"/>	C20180019	汇编语言程序设计...	32	1	

图 4.9 过滤后的选课界面

最后一个 Task Tab 则是任务 Tab, 学生用户可以浏览与自己已选课程有关的任务并完成、提交任务。如图 4.10 所示。

The screenshot shows the student interface with three tabs at the top: SELF, COURSE, and TASK. The TASK tab is selected, indicated by a green underline. A modal window titled "Select a Task" is open, listing various tasks assigned to the student. The tasks include: 模拟电子技术 (二) : W201800021, scores: no score; 模拟电子技术 (二) : W201800062, scores: no score; 模拟电子技术 (二) : W201800059, scores: no score; 模拟电子技术 (二) : W201800174, scores: no score; 数字电路与逻辑设计 (一) : W201800018, scores: no score; 数字电路与逻辑设计 (一) : W201800044, scores: no score; 数字电路与逻辑设计 (一) : W201800169, scores: no score; 数字电路与逻辑设计 (一) : W201800182, scores: no score; 汇编语言程序设计: W201800033, scores: no score; 汇编语言程序设计: W201800061, scores: no score; 汇编语言程序设计: W201800077, scores: no score; 汇编语言程序设计: W201800184, scores: no score; JAVA语言程序设计: W201800047, scores: no score. There is also a "SUBMIT" button at the bottom of the modal.

图 4.10 学生界面任务 tab

教师界面与学生界面格式基本相同，但功能有所不同。此处不再展示重复的界面。教师可以为课程添加任务，其界面如图 4.11 所示。

The screenshot shows the teacher interface with three tabs at the top: SELF, COURSE, and TASK. The SELF tab is selected, indicated by a blue underline. A modal window titled "Add a Task" is open, prompting the teacher to "Select a course:" and "What's the question?". A dropdown menu shows "C201800033: 软件工程". At the bottom of the modal are "CANCEL" and "ADD" buttons. In the background, there are sections for "Personal Information" (Name: 邓心进, User: T20180001, Id: T20180001) and "Teaching Courses" (C201800033: 软件工程). Below these, the "Assigned Tasks" section lists: 软件工程: W20180011 and 软件工程: W20180012. A "ADD A TASK" button is located at the bottom left, and a circular refresh icon is at the bottom right.

图 4.11 添加任务界面

教师的任务列表与学生有所不同，使用了两级列表的方式展现了所有布置的任务以及对应的学生，如图 4.12 所示。

The screenshot shows a user interface for managing tasks. At the top, there are three tabs: 'SELF', 'COURSE', and 'TASK'. The 'TASK' tab is selected, indicated by a teal background and white text. Below the tabs is a search bar with placeholder text 'Select a Task' and a dropdown menu showing 'Software Engineering' tasks: 'W201800114, type: exam' and 'W201800124, type: task'. The main content area is divided into two columns. The left column, under the 'COURSE' tab, is titled 'Select a Student' and lists student records from S201800072 to S201800677, each with a name and score. The right column, under the 'TASK' tab, has a header 'LIST' with a minus sign, a plus sign, and a 'GRADE' button. There is also a small input field with the value '0'.

图 4.12 教师的 Task Tab

对于管理员而言，其界面只有 2 个 tab，主界面如图 4.13 所示。

The screenshot shows the main administrative interface with three tabs: 'INFORMATION' (selected), 'APPLICATIONS', and 'MANAGEMENT'. The 'INFORMATION' tab displays student, course, and teacher data in three separate tables. Each table has a header and a list of entries. A purple circular icon with a refresh symbol is positioned at the bottom center of the page.

INFORMATION		
Students	Courses	Teachers
S201800001 李德彧 S201800002 刘楼林 S201800003 朱增晶 S201800004 曹德余 S201800005 黄小华 S201800006 王耀健 S201800007 顾云芳 S201800008 吴思玉 S201800009 张志双 S201800010 韩牧盛 S201800011 罗福晶 S201800012 黄定婷 S201800013 谢凤淇 S201800014 李明永 S201800015 袁伶长	C201800001 线性代数（一） C201800002 概率论与数理统计（三） C201800003 复变函数与积分变换 C201800004 人文社科类选修课程 C201800005 C语言程序设计 C201800006 C语言程序设计实验 C201800007 电路理论（五） C201800008 模拟电子技术（二） C201800009 信号与线性系统 C201800010 数字电路与逻辑设计（一） C201800011 数字电路与逻辑设计实验 C201800012 计算机通信与网络 C201800013 计算机通信与网络实践 C201800014 离散数学（一） C201800015 离散数学（二）	T201800001 邓心进 T201800002 伍民承 T201800003 丘昌以 T201800004 柳文昌 T201800005 凌子之 T201800006 王伟世 T201800007 卞政贵 T201800008 卓仁之 T201800009 皮涛超 T201800010 莫中泽 T201800011 文庆承 T201800012 夏庆承 T201800013 金山茂 T201800014 华轮江 T201800015 符厚翰

图 4.13 管理员主界面

除了主界面用于显示部分表的信息外，Application Tab 用于显示学生和老师的申请，一个空的 Application Tab 如图 4.14 所示。

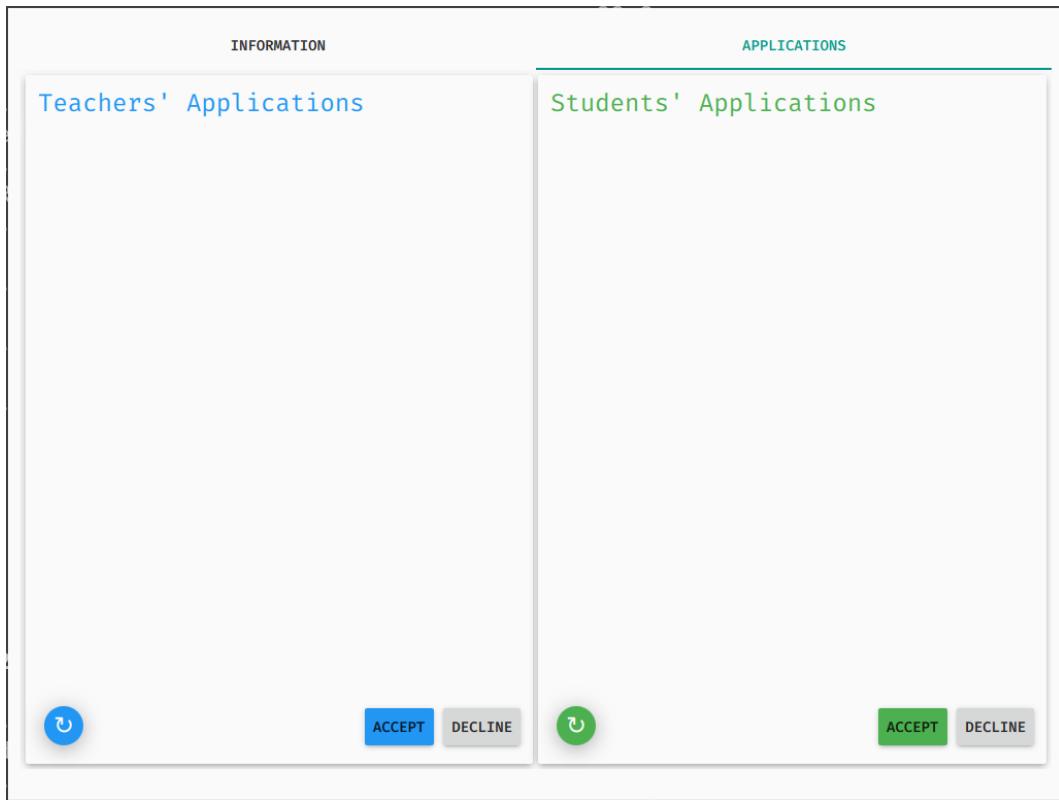


图 4.14 管理员界面的 Application Tab

4.6 系统测试

通过一次完成的对于 3 个角色的模拟，对于系统的功能进行测试：

首先打开服务器，登录学生账号 S20180003，登录后如图 4.15 所示，可以看出此学生只选择了 2 门课程。

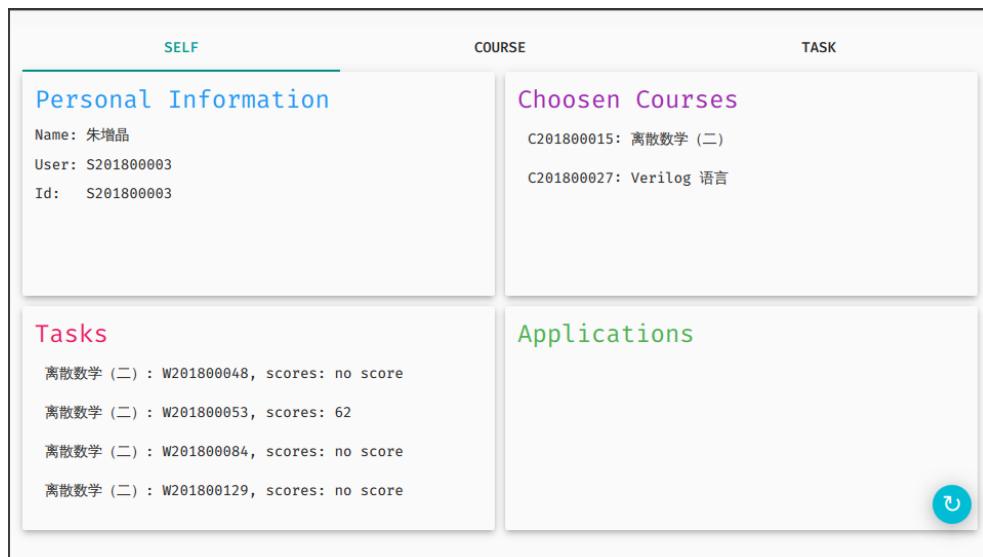


图 4.15 学生初次登录

然后进入选课列表，选择“线性代数”这一门课程，如图 4.16 所示。图示为点击 Register 之前的情形。

SELF		COURSE			TASK	
id	name	span			REFRESH	REGISTER
select	Course Id	Name	Span	Season		
<input checked="" type="checkbox"/>	C201800001	线性代数（一）	40	5		
<input type="checkbox"/>	C201800002	概率论与数理统计...	40	1		
<input type="checkbox"/>	C201800003	复变函数与积分变换	40	5		
<input type="checkbox"/>	C201800004	人文社科类选修课程	160	2		
<input type="checkbox"/>	C201800005	C语言程序设计	48	3		
<input type="checkbox"/>	C201800006	C语言程序设计实验	32	1		
<input type="checkbox"/>	C201800007	电路理论（五）	64	4		
<input type="checkbox"/>	C201800008	模拟电子技术（二）	48	1		
<input type="checkbox"/>	C201800009	信号与线性系统	40	4		
<input type="checkbox"/>	C201800010	数字电路与逻辑设...	56	1		
<input type="checkbox"/>	C201800011	数字电路与逻辑设...	16	5		
<input type="checkbox"/>	C201800012	计算机通信与网络	40	4		
<input type="checkbox"/>	C201800013	计算机通信与网络...	32	4		
<input type="checkbox"/>	C201800014	离散数学（一）	40	6		
<input checked="" type="checkbox"/>	C201800015	离散数学（二）	40	3		

图 4.16 学生选课

这时，登录任意一个管理员的界面(此处选择 M201800001)，进入 Application Tab，可以看到学生的选课申请如图 4.17 所示。若已经提前登录，则不会显示学生成的申请，需要手工点击刷新按钮从服务器重新同步数据。

INFORMATION		APPLICATIONS	
Teachers' Applications		Students' Applications	
<input type="checkbox"/>	A201800001	Register	C201800001
ACCEPT	DECLINE	ACCEPT	DECLINE

图 4.17 管理员界面中学生的选课申请

同意本选课申请后此申请消失。如果拒绝此申请，申请也会消失，但无法继续进行接下来的实验，因此这里点击 Accept 按钮同意这一申请。在有多个申请的情况下，可以同时同意或拒绝多个申请。这一同意或拒绝的列表将由服务端程序逐个进行处理。

回到学生界面，点击刷新按钮，可以看到申请已被同意（申请列表中），如图 4.18 所示。

SELF	COURSE	TASK
Personal Information	Choosen Courses	
Name: 朱增晶 User: S20180003 Id: S20180003	C20180001: 线性代数 (一) C20180015: 离散数学 (二) C20180027: Verilog 语言	
Tasks	Applications	
线性代数 (一) : W20180090, scores: no score 线性代数 (一) : W201800155, scores: no score 线性代数 (一) : W20180004, scores: no score 线性代数 (一) : W20180055, scores: no score	✓ A20180001 Register C20180001	⟳

图 4.18 申请被同意

进入教师界面，让教师选择教授线性代数这一门课，如图图 4.19 所示。图示为选课之后的情形，可以看到，选课之后复选框消失，阻止了可能的重复选择。实际上，即使重复选择也不会对于数据库的稳定性造成任何影响，只会产生重复的请求，而服务器在处理重复的请求是会将除了第一个之外的请求视作无效请求并拒绝掉。

SELF		COURSE			TASK	
id	name	span			REFRESH	REGISTER
select	Course Id	Name	Span	Season		
<input type="checkbox"/>	C20180001	线性代数 (一)	40	5		
<input type="checkbox"/>	C20180002	概率论与数理统计...	40	1		
<input type="checkbox"/>	C20180003	复变函数与积分变换	40	5		
<input type="checkbox"/>	C20180004	人文社科类选修课程	160	2		
<input type="checkbox"/>	C20180005	C语言程序设计	48	3		
<input type="checkbox"/>	C20180006	C语言程序设计实验	32	1		
<input type="checkbox"/>	C20180007	电路理论 (五)	64	4		
<input type="checkbox"/>	C20180008	模拟电子技术 (二)	48	1		
<input type="checkbox"/>	C20180009	信号与线性系统	40	4		
<input type="checkbox"/>	C20180010	数字电路与逻辑设...	56	1		
<input type="checkbox"/>	C20180011	数字电路与逻辑设...	16	5		
<input type="checkbox"/>	C20180012	计算机通信与网络	40	4		
<input type="checkbox"/>	C20180013	计算机通信与网络...	32	4		
<input type="checkbox"/>	C20180014	离散数学 (一)	40	6		
<input type="checkbox"/>	C20180015	离散数学 (二)	40	3		

图 4.19 教师选课

同样的，由任意一个管理员批准后教师就能够教授这一门课程了。此时回到教师的主界面，点击刷新按钮，同步服务器数据，然后点击 Add Task，为线性代数添加一个任务，如图 4.20 所示。

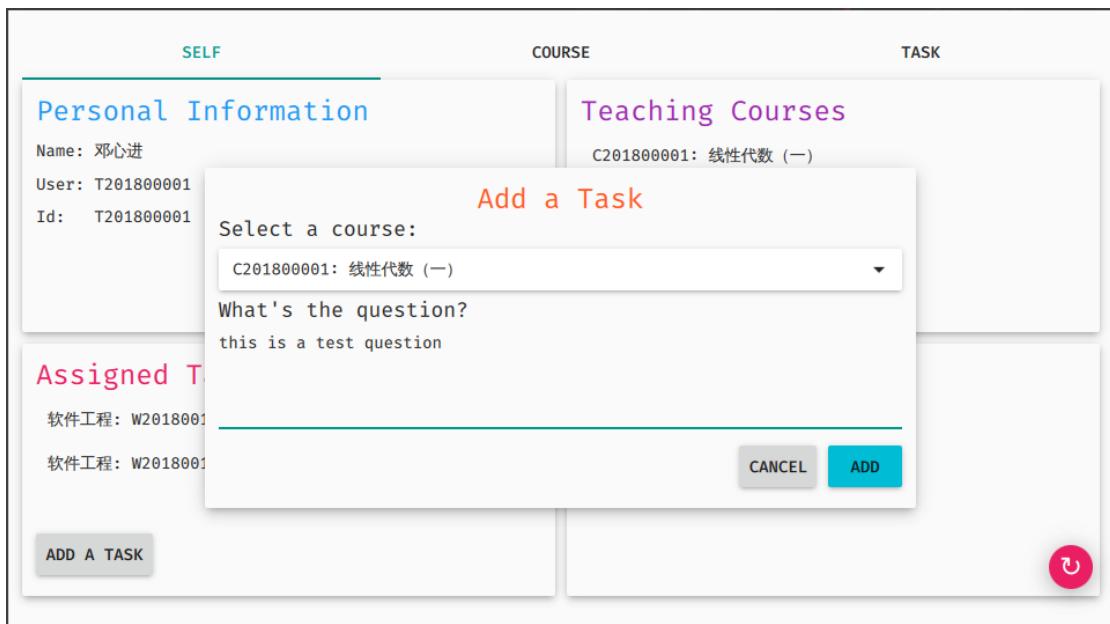


图 4.20 为课程添加任务

添加完成后，学生可以在自己的 Task 面板中看到并完成这一任务，如图 4.21 所示。

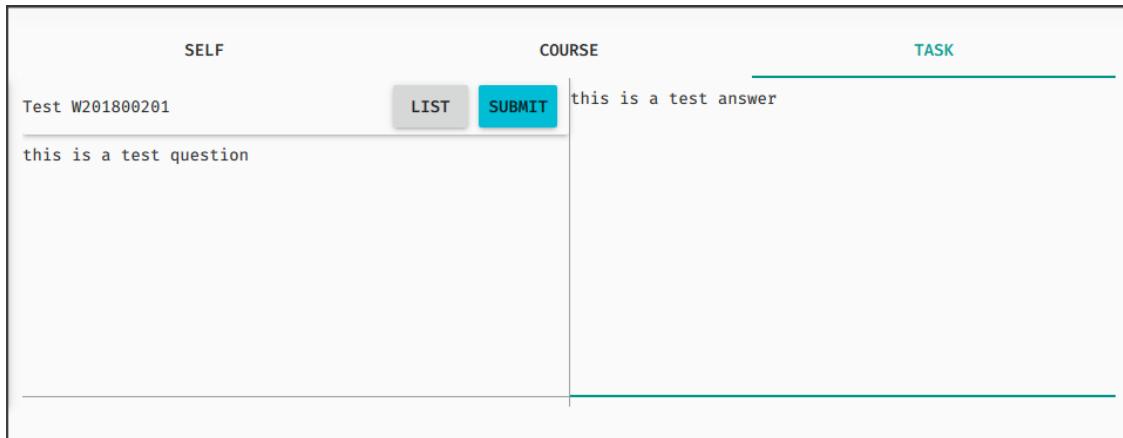


图 4.21 学生完成任务

完成后，教师可以在 Task 页面看到学生的答案并打分，如图 4.22。



图 4.22 教师为学生的答案打分

最后，在教师打分后，学生在自己的主页面可以看到这一门课程的分数，如图 4.23，此时已无法再次修改答案。

The screenshot shows a student's main page with three tabs at the top: SELF, COURSE, and TASK. The SELF tab is active, displaying 'Personal Information' with fields: Name: 朱增晶, User: S20180003, and Id: S20180003. The COURSE tab shows 'Choosen Courses' with three entries: C20180001: 线性代数 (一), C20180015: 离散数学 (二), and C20180027: Verilog 语言. The TASK tab shows 'Tasks' with four entries: 线性代数 (一) : W201800201, scores: 90; 线性代数 (一) : W201800090, scores: no score; 线性代数 (一) : W201800155, scores: no score; and 线性代数 (一) : W201800004, scores: no score. The Applications section shows a checked task: ✓ A201800001 Register C20180001. A blue circular refresh icon is in the bottom right corner.

图 4.23 学生可以查看教师的打分

至此，程序的功能测试完成，基本功能正常。

经测试，由于使用了参数化查询，类似”= -1 or 1 = 1; -- ”这样的针对典型数据库注入攻击对于本程序是无效的，又由于客户端无法直接连接数据库，在服务器本身不被攻破的情况下外界是无法知道数据库的内部结构的，从而更进一步的增加了其安全性。

此外，经过压力测试，在主频 2GHz, SATA ssd 的机器上每秒可以接受并预处理约 20000 个来自客户端的请求，平均每秒可以处理约 8000 个请求并返回给客户端。

4.7 系统设计与实现总结

在这次课程设计中，我对于一个 C/S 模式带数据库的学分管理系统进行了完整的设计。完成的主要工作如下：

- (1) 完成了对于数据库结构以及软件的需求分析
- (2) 完成了数据库的表格创建，数据的生成以及导入
- (3) 设计并实现了模块化服务端
- (4) 设计并实现了用户友好的客户端
- (5) 进行了较为完整的功能、性能以及安全性测试。

5 课程总结

这次课程实验共有 3 个任务，难度逐个递增。在第一个实验中，我完成了与数据库有关的基本操作，包括用户、表的创建，选择与更改语句的实现以及对于事务的实现，这为后续的实验打下了坚实的基础。

在第二个实验中，我完成了包括创建触发器在内的权限管理，并对于存储过程进行了初步的试验，更进一步的了解了数据库的各种功能上的特性。

最后，在前两个实验的基础上，完成了综合性极强的第三个实验，从设计到实现再到测试，不仅将数据库的知识运用的淋漓尽致，更是应用了包括软件工程、算法、C++、计算机系统、计算机网络在内的各种知识，结合这所有的知识设计了一个功能完善，性能优越的系统。

这些实验首先让我巩固了数据库的知识，对于数据库的应用从生手达到了熟手，此外，对于系统能力的培养也是十分有帮助的，尤其是第三个系统设计与实现的实验，通过查找资料以及亲身实践，我对于实际的数据库在软件开发中的应用有了更为深入的了解。

在这次实验中，给我体会最深的是文档查阅的重要性。许多的接口查询以及错误原因的排查都是通过文档解决的，文档能够对于一个系统有详细的描述，从而让人易于使用，这也是在实现系统设计实验时在代码中嵌入了详细的多性格呢文档的原因。此外，细心和耐心也是十分重要的。在发生错误时，这两项是排除错误的必要条件。

总体来说，我对于这一次课程实验的实现效果还是十分满意的，在有限的时间内三个实验均达到了较为理想的结果。不过我希望实验时间能够更为充足，过大的课程压力使得实验时间过短，导致综合实践任务中的一些可使程序跟完整的功能没有完成，一些可以使系统性能更为高效的优化没有实现，不得不说有一些小小的遗憾。不过总体上，这次实验给我带来的收获是十分丰富的。

附录

参考文献

- [1] 萨师煊, and 王珊. 数据库系统概论. Vol. 2. No. 000. 高等教育出版社, 2000.
- [2] 崔巍, and 数据库. 数据库系统及应用. Vol. 1. 高等教育出版社, 1999.
- [3] Prata, Stephen. *C++ primer plus*. Sams Publishing, 2002.
- [4] 齐治昌, et al. 软件工程 (第二版). 高等教育出版社, 2004.
- [5] Bartholomew, Daniel. *MariaDB cookbook*. Packt Publishing Ltd, 2014.
- [6] MySQL, A. B. "MySQL reference manual." (2001).