华中科技大学

数据库系统原理实践报告

专　　业：	计算机科学与技术专业
班　　级：	CS2106
学　　号：	U202115514
姓　　名：	杨明欣
指导教师：	丁晓峰

分数	
教师签名	

2023 年 6 月 20 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	
6	

总分	

目　录

1 课程任务概述

“数据库系统原理实践”是配合“数据库系统原理”课程独立开设的实践课，注重理论与实践相结合。本课程以 MySQL 为例，系统性地设计了一系列的实训任务，内容涉及以下几个部分：

1) 数据库、表、索引、视图、约束、存储过程、函数、触发器、游标等数据对象的管理与编程；
2) 数据查询，数据插入、删除与修改等数据处理相关任务；
3) 数据库的安全性控制，完整性控制，恢复机制，并发控制机制等系统内核的实验；
4) 数据库的设计与实现；
5) 数据库应用系统的开发(JAVA 篇)。

实验环境为 Linux 操作系统下的 MySQL 8.0.28（主要为 8.028 版本，部分关卡使用 8.022 版本，使用中基本无差别）。在数据库应用开发环节，使用 JAVA 1.8。

头歌上将执行内容分为了 15 个实训关卡：实训 1-数据库与表的创建，实训 2-表的修改，实训 3、4-数据查询，实训 5-数据修改、插入和删除，实训 6-视图，实训 7-存储过程与事务，实训 8-触发器，实训 9-用户自定义函数，实训 10-安全性控制，实训 11-并发控制与事务的隔离级别，实训 12-备份、日志与数据恢复，实训 13 数据库设计与实现，实训 14-java 语言对数据库的应用开发，实训 15-索引 B+树实现；

实验总共包括 73 关卡，全部完成则获得头歌平台总分 161 分，不要求所有关卡都完成，最终程序检查满分只计头歌平台中的 100 分，即有些关卡可以自行选择跳过不做，但是实践环节的 2 级标题子任务的前 5 个以及第 13 个子任务（2.1 至 2.5 子任务、2.13 子任务）不能整体跳过（至少要完成其中每个二级子任务中的一个关卡）。

实验包含以下相关材料：

MYSQL 手册：https://dev.mysql.com/doc/

JAVA 手册：https://docs.oracle.com/javase/8/docs/api/index.html

课程开放资源：https://gitee.com/kylin8575543/db2022-spring

2 任务实施过程与分析

本次实践课程在头歌平台进行，实践任务均在平台上提交代码，所有完成的任务、关卡均通过了自动测评。本次实践最终完成了任务书中的 2.1~2.6 任务，2.7 中的前两个任务，2.8~2.14 任务。下面重点对于已经完成的任务进行系统详细的介绍，其中任务 2.2 和任务 2.3 涉及到的查询任务较多，而且有很多重复的知识点，因此仅选取其中部分进行详细阐释。

2.1 数据库、表与完整性约束的定义(Create)

此部分实验主要考察使用 MYSQL 提供的数据定义语句（DDL）完成创建数据库和基本表、数据完整性约束条件的定义等一系列操作，具体来说就是通过 CREATE 语句实现实验要求的具体内容。此部分实验中通过了 1~6 关所有的关卡。

2.1.1 创建数据库

使用 CREATE DATABASE <数据库名称>进行数据库创建，具体代码如下：

```
CREATE DATABASE beijing2022;
```

2.1.2 创建表及表的主码约束

使用 CREATE TABLE <基本表名称>进行基本表的创建，通过 PRIMARY KEY 列级完整性约束声明主码，具体代码如下：

```
CREATE TABLE t_emp (id int PRIMARY KEY,
name varchar(32),
deptId int,
salary float);
```

2.1.3 创建外码约束(foreign key)

创建列级外码约束时，在列的名称和数据类型后，直接用关键词 references 定义外码约束，指明该外码对应的主码（表名和列名），同时还可以定义（可选）一旦违反参照完整性时，应采取何种应对策略。同样也可以通过 CONSTRAINT 创建表级约束，表级外码约束的好处是可以给约束命名，且支持多属性组合外码。具体代码如下：

```
CONSTRAINT FK_staff_deptNo FOREIGN KEY(deptNo) REFERENCES dept(deptNo)
```

2.1.4 CHECK 约束

创建列级 CHECK 约束时，可以在列定义后 CONSTRAINT <约束名> CHECK(约束条件)；同样在创建表级 CHECK 约束时，可以单独声明 CONSTRAINT <约束名> CHECK(约束条件)。具体代码如下：

```
brand char(10) CONSTRAINT CK_products_brand check(brand in ('A', 'B')),
price int CONSTRAINT CK_products_price check(price > 0)
```

2.1.5 DEFAULT 约束

默认值约束(Default 约束)用于给表中的字段指定默认值，即往表里插入一条新记录时，如果没有给这个字段赋值，DBMS 就会自动赋于这个字段默认值。具体代码如下：

```
mz char(16) default('汉族')
```

2.1.6 UNIQUE 约束

唯一性约束(Unique 约束)用于保证表中字段取值的唯一性。Unique 约束既可以约束表中的单列，也可以约束表中的组合列（多列）。具体代码如下：

```
ID char(18) Unique
```

2.2 表结构与完整性约束的修改(ALTER)

此部分实验主要考察使用 MYSQL 提供的数据定义语句（DDL）完成表结构与完整性约束的修改等一系列操作，具体来说就是通过 DROP 语句、ALTER 语句实现实验要求的具体内容。此部分实验中通过了 1~4 关所有的关卡。

2.2.1 修改表名

ALTER TABLE 语句用于修改由 CREATE TABLE 语句创建的表的结构。可以通过 ALTER TABLE <旧表名> RENAME <新表名>的方式修改表名。具体代码如下：

```
ALTER TABLE your_table RENAME my_table;
```

2.2.2 添加与删除字段

可以通过 ALTER TABLE <表名> DROP [COLUMN] <列名>删除字段，其中关键字 COLUMN 可以省略，具体代码如下：

```
alter table orderDetail drop orderDate;
```

也可以通过 ALTER TABLE <表名> ADD [COLUMN] <列名> <数据类型> [列约束] [FIRST | AFTER 列名]添加字段，具体代码如下：

```
alter table orderDetail add unitPrice decimal(10,2) after quantityOrdered;
```

2.2.3 修改字段

可以通过 RENAME 关键字修改列的名称，具体语法为 ALTER TABLE <表名> RENAME COLUMN <列名> TO <新列名>，具体代码如下：

```
alter table addressBook rename column weixin to wechat;
```

也可以通过 MODIFY 关键字修改其数据类型和约束，具体语法为 ALTER TABLE <表名> MODIFY [COLUMN] <列名> <数据类型> [列约束] [FIRST | AFTER col_name]，在修改数据类型和约束的同时，还可以改变列在表中的位置。具体代码如下：

```
alter table addressBook modify QQ char(12);
```

2.2.4 添加、删除与修改约束

添加主码的语法为 ALTER TABLE <基本表> ADD CONSTRAINT PRIMARY KEY(列名)，删除主码的语法为 ALTER TABLE <基本表> DROP PRIMARY KEY;为基本表添加外码约束和用户自定义约束的基本语法为 ALTER TABLE <基本表> ADD CONSTRAINT (约束条件)。具体代码如下：

```
alter table Staff add constraint primary key(staffNo);
alter table Staff add constraint FK_Staff_dept foreign key(dept) references Dept(deptNo);
alter table Staff add constraint CK_Staff_gender check(gender='F'or gender='M');
```

2.3 数据查询(Select)之一

此部分主要考察对于数据库中表中信息的数据查询操作，主要内容为通过 SELECT 语句实现包括但不限于单表查询、嵌套查询、（多）条件查询、多表连接、对于查询结果的排序、分组、消重及统计、特殊函数、子查询、谓词\关键词、对表达式列命名、衍生表、分表合并等一系列操作。此部分实验中通过了 1~19 关所有的关卡，仅展示部分实验的详细思路。

2.3.1 办理了储蓄卡的客户信息

查询办理了储蓄卡的客户名称、手机号、银行卡号。将 client 和 bank_card

通过 join 进行多表连接，条件是 b_c_id = c_id，然后筛选出办理储蓄卡的客户，具体代码如下：

```
select c_name,c_phone,b_number from client join bank_card on b_c_id = c_id
where b_type='储蓄卡'
order by c_id;
```

2.3.2 商品收益的众数

查询资产表中所有资产记录里商品收益的众数和它出现的次数。对 property 表使用 group by 子句对进行分组统计，并且使用 having 子句选出其中元组个数最大的分组，在这里 having 子句有两种思路，第一种思路为选择组内元组个数大于等于所有分组元组个数的分组，使用关键字 ALL 和子查询，具体代码如下：

```
HAVING COUNT(*) >= (SELECT MAX(num) FROM (SELECT COUNT(*) AS num
FROM property GROUP BY pro_income) AS t);
```

另一种思路为选择组内元组数目大于等于最大的元组个数，使用 MAX 函数和子查询，具体代码如下：

```
HAVING COUNT(*) >= (SELECT MAX(num) FROM (SELECT COUNT(*) AS num
FROM property GROUP BY pro_income) AS t);
```

2.3.3 持有两张信用卡的用户

查询在本行持有两张及以上信用卡的客户信息。首先在子查询中对于 b_c_id 和 b_type 按照分组查询，选择元组数大于 1 的组别，通过(c_id, '信用卡')外部筛选，通过 c_id 在 client 查找相关信息。具体代码如下：

```
SELECT c_name, c_id_card, c_phone
FROM client
WHERE (c_id, '信用卡') in (
    SELECT b_c_id, b_type
    FROM bank_card
    GROUP BY b_c_id, b_type
    HAVING COUNT(*) > 1
);
```

2.3.4 投资总收益前三名的客户

查询投资总收益前三名的客户。对于 client 和 property 进行等值连接，然后选取其中 pro_status 状态为“可用”的元组，之后按照 c_id 分组，使用 SUM 函数统计其中的 pro_income，然后通过 rank 函数进行对 total_income 降序排序，最后获取其中的前三个元组，即总收益前 3 名的客户。具体代码如下：

```
Select c_name, c_id_card, sum(pro_income) as total_income
from client, property
where pro_c_id=c_id and pro_status="可用"
group by c_id
having rank() over(order by total_income desc) > 3;
```

2.3.5 客户理财、保险与基金投资总额

查询客户理财、保险、基金投资金额的总和，并排序。此问题的查询通过派生表的统计、多表合并统计总金额。具体来说，首先通过分别对于理财产品表(finances_product)、保险表(insurance)和基金表(fund)与资产表(property)进行连接，通过 SUM(商品数量*该产品每份金额)公式进行计算不同用户每类商品的投资总金额，最终通过 union all（此处同样可以使用 union）将三个派生表进行合并，然后再与客户表(client)进行 left outer join 连接，获得可以进行投资金额查询的派生表。

然后进行投资金额的统计。按照用户 c_id 进行分组，统计每个用户的投资总金额。具体代码省略。

2.3.6 第 N 高问题

查询每份保险金额第 4 高保险产品的编号和保险金额。通过子查询对于保险表(insurance)按照金额进行排序，考虑到可能会有相同金额的保险产品，因此通过 dense_rank() over(order by 列名)进行排名的获取，得到派生表，最终通过设定 rank 为 4 的保险进行获取，按照 i_id 进行排序获取最终结果。

```
SELECT i_id, i_amount
FROM (
    SELECT i_id, i_amount, dense_rank() OVER (ORDER BY i_amount DESC) AS rk
    FROM insurance
```

```
    ) t
WHERE rk = 4
ORDER BY i_id;
```

2.3.7 持有完全相同基金组合的客户

查询持有完全相同基金组合的客户。首先需要构造两张相同的客户基金组合派生表，然后进行元组的选取。

构造客户基金组合派生表时，主要通过使用聚集函数 GROUP_CONCAT 进行组内基金号的合并，命名为 f_id，生成客户基金的派生表，分别定义其为两张表 t1 和 t2。进行元组选取时，则通过进行 f_id 的比较，按照选取前用户的 id 小于后用户 id 的原则进行元组的选取，最终按照用户 id 进行升序排列，具体代码省略。

2.3.8 购买基金的高峰期

查询 2022 年 2 月购买基金的高峰期，如果连续三个交易日，投资者购买基金的总金额超过 100 万，则称这连续的几日为投资者购买基金的高峰期。这个实验的难度比较大，关键在于如何通过合适的算法进行连续三个交易日的选取。最后确定本实验的算法为通过对于日期进行工作日时间的转换，获取新的变量为该日期对应 2023 年第几工作日 wk，然后对于再这些工作日中交易日内金额大于 100 万元的元组进行 RANK()排序 rk，此时计算 wk 和 rk 的差值，易得差值相同的交易日为连续交易日。

最终通过 COUNT(*) OVER (PARTITION BY wk- rk) cnt 进行计算，获取其中 cnt 大于 3 的元组，即为最终购买基金的高峰期。具体代码省略。

2.4 数据查询(Select)之二

此部分主要考察对于数据库中表中信息的数据查询操作，主要内容为通过 SELECT 语句实现包括但不限于嵌套查询、子查询、关键函数使用等一系列操作。此部分实验中通过了 1~6 关所有的关卡，仅展示部分实验的详细思路。

2.4.1 查询销售总额前三的理财产品

查询销售总额前三的理财产品。此实验整体流程为，首先将 pro_purchase_time 转换成年份，然后依照年份和产品编号 p_id 进行分组，分别

计算每一年每个理财产品的销售总额 sumamount，最后通过 RANK() OVER(PARTITION BY pyear ORDER BY sumamount DESC)进行分年份排序，获取其中 2010 年和 2011 年排序在前三名的理财产品，即为最终结果。具体代码省略。

2.4.2 查询购买了所有畅销理财产品的客户

查询购买了所有畅销理财产品的客户。此实验整体流程为，首先获得两张派生表，其一为畅销理财产品，由 COUNT(*)大于 2 的理财产品构成，其二为每个客户所购买的理财产品表。

然后，进行逻辑转换，将查询购买了所有畅销理财产品的客户这一逻辑转化为查询所有畅销理财产品没有不在其购买的理财产品表中的客户（通过双重否定表示肯定），通过 NOT EXISTS 和 NOT IN 语句进行实现，具体代码省略。

2.5 数据的插入、修改与删除(Insert,Update,Delete)

此部分主要考察对于数据库中表中数据进行插入、删除与修改操作，主要内容为通过 INSERT、DELETE 和 UPDATE 实现上述操作。此部分实验中通过了 1~6 关所有的关卡。

2.5.1 插入多条完整的客户信息

使用 INSERT INTO <表名> VALUES (元组值)进行数据插入。具体代码如下：

```
INSERT INTO client
VALUES
(1,'林惠雯','960323053@qq.com','411014196712130323','15609032348','Mop5UPkl');
```

2.5.2 插入不完整的客户信息

使用 INSERT 插入不完整信息要声明列名，具体代码如下：

```
INSERT INTO client (c_id, c_name, c_phone, c_id_card, c_password)
VALUES(33,'蔡依婷','18820762130','350972199204227621','MKwEuc1sc6');
```

2.5.3 批量插入数据

对于结构完全相同的数据可以通过子查询方式进行插入。具体代码如下：

```
INSERT INTO client
select * from new_client;
```

2.5.4 删除没有银行卡的客户信息

使用 DELETE FROM <表名> <子查询>进行数据删除。具体代码如下：

```
DELETE FROM   client
where not exists (select * from bank_card where client.c_id = bank_card.b_c_id);
```

2.5.5 冻结客户资金

使用 UPDATE 语句，WHERE 子句使用嵌套子查询。具体代码如下：

```
update property
set pro_status = '冻结'
where pro_c_id in (
    select c_id
    from client
    where c_phone = '13686431238'
);
```

2.5.6 连接更新

使用 UPDATE 语句，通过 set 子句，将 pro_c_id 和 c_id 进行等值连接，通过 c_id_card 更新对应的 pro_id_card。具体代码如下：

```
update property
set pro_id_card = (
    select c_id_card from client where pro_c_id = c_id
);
```

2.6 视图

此部分主要考察对于数据库进行视图的创建、基于视图的查询操作等，主要内容为通过 CREATE VIEW 进行视图创建、SELECT 基于视图进行查询。此部分实验中通过了 1~2 关所有的关卡。

2.6.1 创建所有保险资产的详细记录视图

使用 CREATE 语句实现视图的创建，具体实现语句为 CREATE VIEW <视图名> [<列名>[，<列名>，…]] AS <子查询> [WITH CHECK OPTION];。在本次实验中为对于 client 表、property 表和 insurance 表进行等值连接，将其中所需要的

属性投影到新的外模式中。具体代码如下：

```
CREATE VIEW v_insurance_detail AS
SELECT c_name,c_id_card,i_name,i_project,pro_status,pro_quantity,i_amount,i_year,
    pro_income,pro_purchase_time
FROM client,property, insurance
WHERE
    c_id = pro_c_id
    AND pro_type = 2
    AND pro_pif_id = i_id;
```

2.6.2 基于视图的查询

基于视图的查询与基于基本表的查询没有太大的区别，具体代码如下：

```
SELECT
    c_name,
    c_id_card,
    SUM(pro_quantity * i_amount) AS insurance_total_amount,
    SUM(pro_income) AS insurance_total_revenue
FROM v_insurance_detail
GROUP BY c_id_card
ORDER BY insurance_total_amount desc;
```

2.7 存储过程与事务

此部分主要考察对于变量的定义和赋值、复合语句与流程控制语句、存储过程的定义、存储过程的创建和调用、存储过程的查询和删除以及游标的定义和使用等，体现了 SQL 语句模块化的编程思想。具体来说就是通过三种具体的控制结构构建存储过程，包括使用流程控制语句的存储过程、使用游标的存储过程和使用事物的存储过程。此部分实验中通过了 1~2 关的关卡。

2.7.1 使用流程控制语句的存储过程

创建一个存储过程，向表 fibonacci 插入斐波拉契数列的前 n 项。首先使用 declare 语句定义变量，并赋予默认值或初始值，未赋默认值则初始值为 null，其

定义方式与其他语言定义变量的形式类似，复制时则通过 SET 语句进行变量的赋值。

接下来则通过具体的复合语句和流程控制语句实现斐波拉契数列的构造算法，其中流程控制语句包括判断语句，具体形式为 IF <条件判断> THEN [陈述性语句] END IF，也包括循环语句，具体形式为 WHILE <条件判断> DO [陈述性语句] END WHILE。对应斐波那契数列问题，则为在项数大于 0 时，需要把其中斐波那契数列第一项 (0,0) 插入进去，在项数大于 1 时，需要把其中斐波那契数列第二项 (1,1) 插入进去，进一步在项数更多的时候，可以通过斐波那契数列的逻辑思路，在插入某一项时，a 为该项的前第二项，b 为该项的前第一项，i 为依次递增的序号，则在此插入斐波那契数列的这一项(i,a+b),然后依次更新 a、b、i 三项，直到满足需要插入的项数时停止插入。

2.7.2 使用游标的存储过程

使用游标编程存储过程为医院的某科室排夜班值班表。在这个实验中，游标的定义和使用尤其重要。其中，SQL 操作都是面向集合的，即操作的对象以及运算的结果均为集合，但有时候，我们需要一行一行地处理数据，这就需要用到游标(CURSOR)，它相当于一个存储于内存的带有指针的表，每次可以存取指针指向的一行数据，并将指针向前推进一行。游标的数据通常是一条查询语句的结果。对游标的操作一般要用循环语句，遍历游标的每一行数据，并将该行数据读至变量，再根据变量的值进行所需要的其它操作。

在此实验中，首先，我们需要定义变量，通过 DECLARE 进行变量的定义，使得他们能够用来进一步设计算法逻辑。进一步进行游标的定义，在此问题中则需要定义两个游标，一个游标 nur_name 用来遍历护士列表，另一个游标 doc_type 用来遍历包含科室主任和普通医生的医生列表，因为医生列表有等级区分，需要在获取医生姓名的同时需要获取医生的等级，为下一步的分析做准备。

然后进行迭代分析，迭代考虑每一个日期的排班安排。此问题中尤其需要考虑的就是周六、周日和周一三天的医生安排，因此产生三种可能情况：（1）当前排班医生为科室主任且时间为周六或者周日时，则需要将科室主任放入临时变量中，同时设置标志变量 tp 表示其为需要进行安排的医生，然后通过医生游标进一步获取下一个医生；（2）当前时间为周一且通过标志变量可以判断其中有临时

的科室主任需要安排时，则优先将科室主任安排为医生；（3）其余情况下，仍然正常通过游标获取医生。

最后，在定义的游标遍历结束后不能循环遍历，因此需要通过 DONE 信号判断是否遍历结束，如果遍历结束，则需要重启游标，才能进一步从头开始进行遍历操作。

2.8 触发器

此部分主要考察 MySQL 的流程控制编程、触发器的基本知识、触发器的创建、触发触发器的时机、触发触发器的事件以及触发器内的特殊表等。具体来说就是通过创建触发器进行合法性检查，要求在数据的 INSERT、UPDATE 和 DELETE 的过程中，对于数据的完整性进行检查，依照检查结果进行反馈，包括正确执行或者返回出错信息。此部分实验中通过了第 1 关所有的关卡。

2.8.1 为投资表 property 实现业务约束规则-根据投资类别分别引用不同表的主码

在本部分，主要为插入数据构建触发器，对应的基本原则是 insert 触发器可以访问 new 表,其内容为 insert 的新数据，因此可以通过 new 表的数据进行信息是否出错的判断。

首先，需要创建触发器结构，在 INSERT 插入数据之前对于新数据进行合法性的判断。

其次则需要考虑错误的具体情况，在此部分为 4 中错误情况：

（1） pro_type 数据不合法时，即插入的资产类型不在已有的资产类列表里面时，显示: type x is illegal!

（2） pro_type = 1,但 pro_pif_id 不是 finances_product 表中的某个主码值，显示:finances product #x not found!

（3） pro_type = 2,但 pro_pif_id 不是 insurance 表中的某个主码值，显示:insurance #x not found!

（4） pro_type = 3,但 pro_pif_id 不是 fund 表中的某个主码值，显示:fund #x not found!

在过程中，根据不同的情况获取报错信息之后，通过 SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;进行报错信息的抛出。

2.9 用户自定义函数

此部分主要考察 MySQL 中函数的定义以及在 SELECT 语句中应用自定义函数等。具体来说就是通过定义特定的模块化函数，满足用户特定的需求，在之后的多次调用中可以发挥很大的作用。此部分实验中通过了第 1 关所有的关卡。

2.9.1 创建函数并在语句中使用它

编写一个依据客户编号计算其在本金融机构的存储总额的函数,并在 SELECT 语句使用这个函数。使用 CREATE FUNCTION function_name([para data_type[,...]]) returns data_type begin function_body; return expression; end 语句进行用户函数的自定义。在此部分函数定义的具体代码如下：

```
create function get_deposit(client_id int)
returns numeric(10,2)
begin
    return ( SELECT SUM(b_balance) FROM bank_card WHERE b_type = '储蓄卡'
            GROUP BY b_c_id HAVING b_c_id = client_id);
end$$
```

进一步调用自定义函数的方法，与其他函数的调用方法基本相同，具体代码如下：

```
select *
from (select c_id_card, c_name, get_deposit(c_id) total_deposit
      from client) tmp
where total_deposit >= 1000000
order by total_deposit desc;
```

2.10 安全性控制

此部分主要考察 MySQL 的安全控制机制、create user 语句的使用和 rant 和 revoke 语句的使用等。具体来说就是通过自主存取控制方法，通过授予创建用户、特定用户特定的权限和收回特定的权限等，保证数据库安全，防止数据泄露。此部分实验中通过了 1~2 关所有的关卡。

2.10.1 用户和权限

使用 CREATE USER <用户名> identified by <用户登录密码>创建用户，具体代码如下：

```
create user tom identified by '123456';
```

使用 GRANT [SELECT|DELETE|UPDATE|INSERT] <列名> on <表名> to <用户> [with grant option]进行插入、删除、修改或者查询权限的授予。具体代码如下：

```
grant
select (c_name, c_mail, c_phone) on client to tom with grant option;
```

使用 REVOKE 权限[,权限]... on 数据库对象 from user|role[,user|role]...将权限从用户或者角色中收回。具体代码如下：

```
revoke
select on bank_card from Cindy;
```

2.10.2 用户、角色与权限

创建角色，授予角色一组权限，并将角色代表的权限授予指定的一组用户。使用 CREATE ROLE <角色名>，具体代码如下：

```
create role client_manager;
```

将权限授予角色的方式与授予角色的方式相似。具体代码如下：

```
grant
select, insert, update on client to client_manager;
```

使用 GRANT <角色> TO <用户>将角色权限授予用户。具体代码如下：

```
grant client_manager to tom, jerry;
```

2.11 并发控制与事务隔离等级

此部分主要考察并发操作可能产生的数据不一致性、MySQL 的事务隔离级别以及隔离性、一致性、并发性的关系等。具体来说就是掌握在并发操作中可能产生的数据不一致性，包括丢失修改、读脏、幻读、不可重复读等，以及通过共享锁和写锁解决上述问题，保证数据库隔离特性。此部分实验中通过了 1~6 关所有的关卡。

2.11.1 并发控制与事务的隔离级别

使用 set session transaction isolation level <隔离等级>语句设置事务的隔离等级。事务隔离级别从低到高分以下四级，读未提交（READ UNCOMITTED）、读已提交（READ COMMITTED）、可重复读（REPEATABLE READ）、可串行化（SERIALIZABLE）。然后通过 start transaction;开启事务。具体代码如下：

```
set session transaction isolation level read uncommitted;
start transaction;
insert into dept(name) values('运维部');
rollback;
```

2.11.2 读脏

读脏(dirty read)，或者又叫脏读，是指一个事务(t1)读取到另一个事务(t2)修改后的数据，后来事务 t2 又撤销了本次修改(即事务 t2 以 roll back 结束)，数据恢复原值。这样，事务 t1 读到的数据就与数据库里的实际数据不一致，这样的数据被称为“脏”数据，意即不正确的数据。只有一种隔离级别可能会产生读脏，即为读不提交。

其中主要通过 set @n = sleep(时间)进行时间上的控制，以事务 1 为例，其事务的具体代码如下：

```
set @n = sleep(1);
select tickets from ticket where flight_no = 'CA8213';
commit;
```

2.11.3 不可重复读

不可重复读(unrepeatable read)，是指一个事务(t1)读取到某数据后，另一个事务(t2)修改了该，事务 t1 并未修改该数据，但当 t1 再次读取该数据时，发现两次读取的结果不一样。不可重复读产生的原因，是事务 t1 的两次读取之间，有另一个事务修改了 t1 读取的数据。有两种级别可以发生不可重复读的现象。具体来说，本实验需要通过更加精细化的时间控制，实现对于两个事务不可重复读现象的重现。具体代码省略。

2.11.4 幻读

幻读是指一个事务(t1)读取到某数据后，另一个事务(t2)作了 insert 或 delete

操作，事务 t1 再次读取该数据时，魔幻般地发现数据变多了或者变少了(记录数量不一致)；而不可重复读限指事务 t2 作了 update 操作，致使 t1 的两次读操作读到的结果(数据的值)不一致。幻读与不可重复读之间的区别就是幻读是发现数据莫名其妙地增加或减少，而不可重复读现象是数据在两次读取时，发现读取的内容不一致的现象。幻读产生的原因，是事务 t1 的两次读取之间，有另一个事务 insert 或 delete 了 t1 读取的数据集。具体代码省略。

2.11.5 主动加锁保证可重复读

MySQL 提供了主动加锁的机制，使得在较低的隔离级别下，通过加锁，以实现更高级别的一致性。SELECT 语句支持 for share 和 for update 短语，分别表示对表加共享(Share)锁和写(write)锁，共享锁也叫读锁，写锁又叫排它锁。具体添加共享锁的代码如下：

```
select tickets from ticket where flight_no = 'MU2455' for share;
```

2.11.6 可串行化

多个事务并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同。两个事务 t1,t2 并发执行，如果结果与 t1→t2 串行执行的结果相同，或者与 t2→t1 串行执行的结果相同，都是正确的(可串行化的)。

选择除 serializable(可串行化)以外的任何隔离级别，保证两个事务并发执行的结果是可串行化的。需要通过时间控制，交错执行程序，实现可串行化调度。具体代码省略。

2.12 备份+日志：介质故障与数据库恢复

此部分主要考察 MySQL 的恢复机制、MySQL 提供的备份与恢复工具等。具体来说就是在具体的生产环境中，难免出现数据库遭到破坏、存储介质故障等问题，和大多数 DBMS 一样，MySQL 利用备份、日志文件实现恢复。此部分实验中通过了 1~2 关所有的关卡。

2.12.1 备份与恢复

使用备份工具 mysqldump 进行数据库备份，备份语句为 mysqldump -h127.0.0.1 -uroot -p123123 [options] --databases db_name。具体来说，对数据库 residents 作海量备份，备份至文件 residents_bak.sql。具体代码如下：

```
mysqldump -h127.0.0.1 -uroot --databases residents >residents_bak.sq1
```

然后，在数据库发生损坏时，可以使用 mysql 执行 SQL 脚本，还原或创建新库。具体来说，利用备份文件 residents_bak.sql 还原数据库。具体代码如下：

```
mysql -h127.0.0.1 -uroot <residents_bak.sq1
```

2.12.2 备份+日志：介质故障的发生与数据库的恢复

在生产环境中，数据库文件、备份文件、日志文件都不会存放在同一介质上，甚至都不会存放在同一地理空间。为的就是灾难发生后，有机会恢复数据。本实验要求模拟介质故障的发生，以及如何利用备份和备份之后的日志恢复数据库。可以通过--flush-log 参数进行日志文件的创建和更新。具体代码如下：

```
mysqldump -h127.0.0.1 -uroot --flush-logs --databases train >train_bak.sql
```

然后，在介质故障发生之后，首先用 mysql 执行 SQL 脚本通过备份数据 train_bak.sql 还原数据库。进一步使用日志进行文件的恢复，当日志文件达到一定规模时，MySQL 还会自动开启新日志文件。具体代码如下：

```
mysql -h127.0.0.1 -uroot <train_bak.sql

mysqlbinlog --no-defaults log/binlog.000018 | mysql -uroot
```

2.13 数据库设计与实现

此部分主要考察数据库设计的阶段和每阶段的任务、概念模型、逻辑模型及其与概念模型的关系、在 DBMS 中的物理实现等。具体来说就是在整体数据库的设计流程中。首先在需求分析的基础上，设计概念模型（主要为 E-R 图方法），进一步转换成逻辑模型，最后进行物理模型的构建，完成整个数据库的设计。此部分实验中通过了 1~3 关所有的关卡。

2.13.1 从概念模型到 MySQL 实现

此实验主要通过对于建模好的概念模型，主要是用 ER 图描述数据及数据间的关系，进行关系模型的转换，进而构建基本表。此实验针对的机票订票系统包括用户、旅客、机场、航空公司、民航飞机、航班常规调度表、航班表、机票八个实体以及各种关系。同时，为了保证实体间关系的正确表达，可以将其中的 ticket、flight、flightschedule 等视作关系，将其中的关联关系作为其外码进行管理。

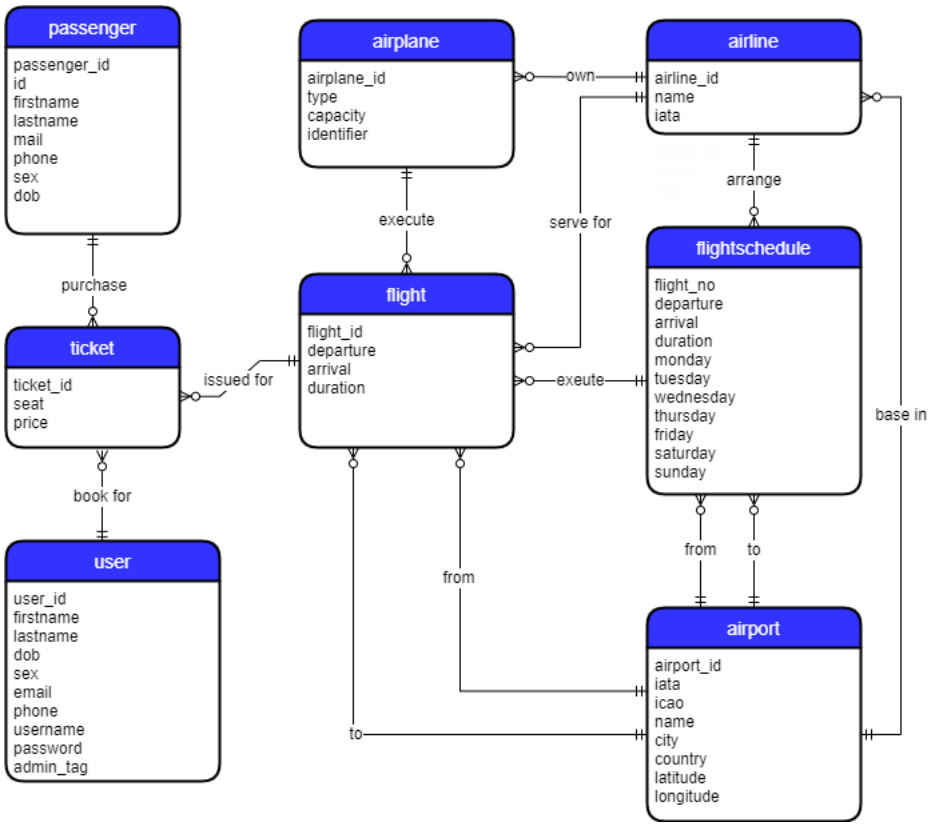

图 2.1 机票订票系统概念模型

除此之外，建表时也要同时设立完整性约束，包括建库，建表，创建主码，外码，索引，指定缺省，不能为空等约束，所有索引采用 BTREE。在列名与关键字同名的时候，需要将列名添加单引号。

2.13.2 从需求分析到逻辑模型

本实验需要设计一个影院管理系统。影院对当前的放映厅和电影进行排片，顾客到来后，可以购买任一排场的电影票，进入对应放映厅观看。系统主要包括电影(movie)、顾客(customer)、放映厅(hall)、排场(schedule)和电影票(ticket)五个实体，同时也提供了实体间的关系，根据其设计概念模型完成 E-R 图如图 2.2 所示，具体链接为 https://s1.ax1x.com/2023/06/21/pCGkpon.jpg。

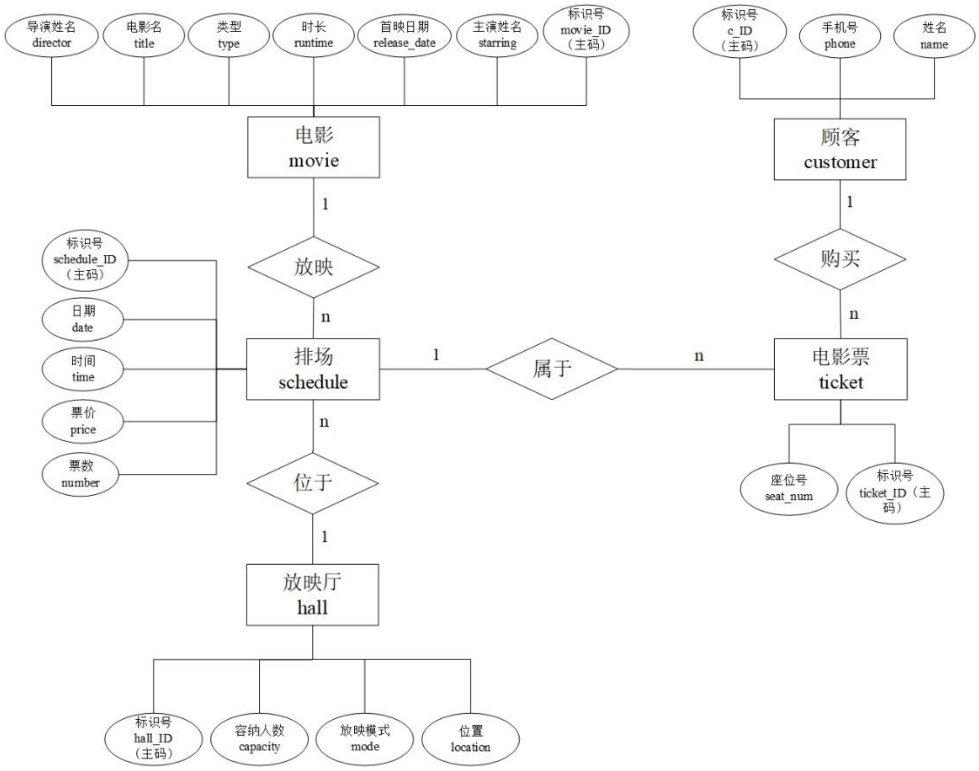

图 2.2 影院管理系统概念模型

根据 E-R 图转换成关系模式如下：

电影(movie)(movie_ID,title,type,runtime,release_date,director,starring);主码:(movie_ID);

顾客(customer)(c_ID,name,phone);主码:(c_ID);

放映厅(hall)(hall_ID,mode,capacity,location);主码:(hall_ID);

排场(schedule)(schedule_ID, date, time, price, number, hall_ID, movie_ID); 主码:(schedule_ID);外码:(hall_ID,movie_ID);

电影票(ticket)(ticket_ID,seat_num,c_ID,schedule_ID); 主码:(ticket_ID); 外码:(c_ID,schedule_ID);

2.13.3 建模工具的使用

此部分需要单独安装 MySQL Workbench，利用 MySQL Workbench 的 forward engineering 功能，将已有的模型文件自动转换成 SQL 脚本。具体来说，将已建模的模型文件 rabc.mwb 导入到 MySQL Workbench 中，如图 2.3 所示。

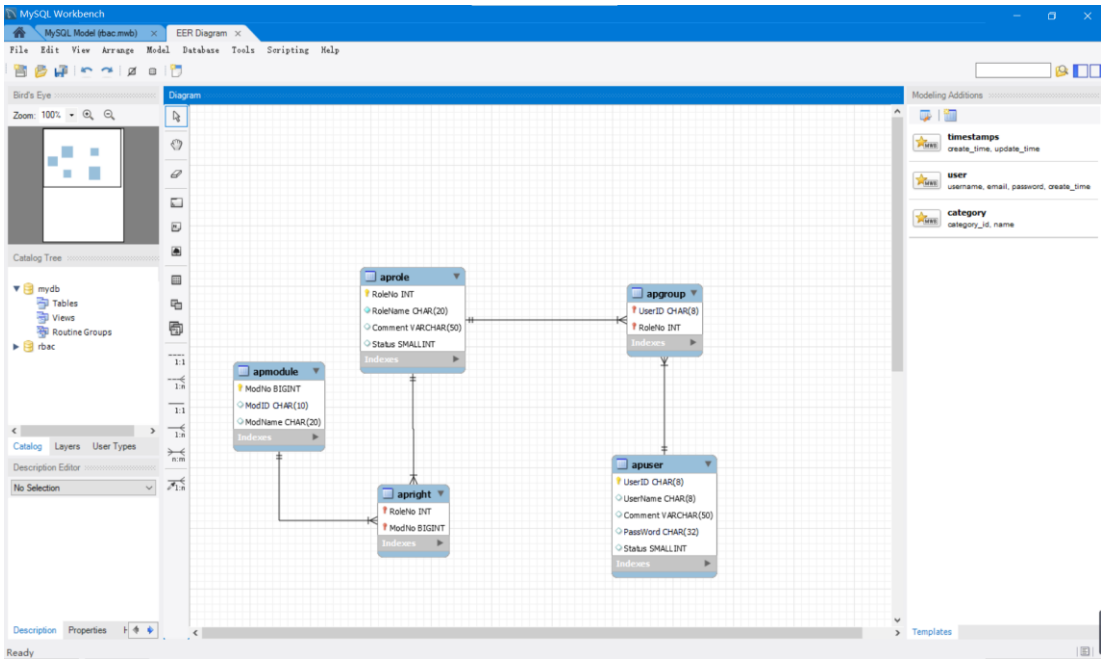

图 2.3 建模文件导入

然后使用菜单栏中的 Database->Forward Engineer 功能，使用 localhost 作为 connection，输入数据库密码则可以完成 SQL 脚本导出，如图 2.4 所示。

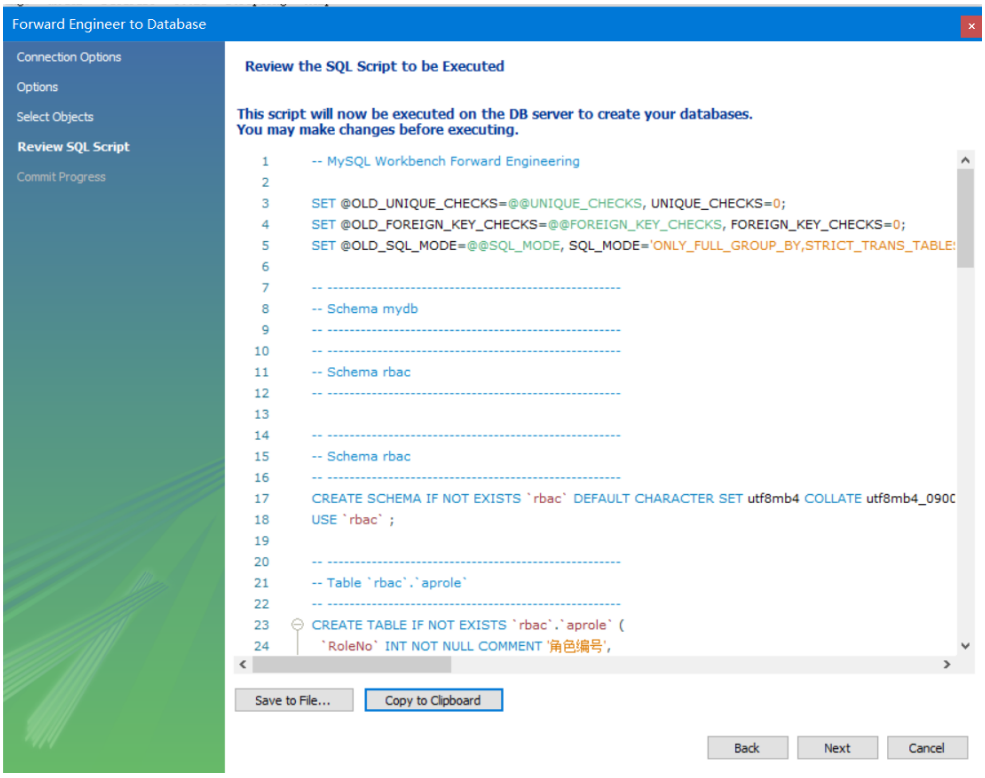

图 2.4 SQL 语句导出

2.13.4 制约因素分析与设计

在进行需求分析时需要充分考虑社会上的需求，在充分考虑的基础上并建立

概念模型，例如在上述图 2.1 的机票订票系统概念模型以及图 2.2 影院管理系统概念模型，都是在充分进行需求分析和考虑的基础上实现的。在购价数据库的逻辑模型和物理模型时，则需要考虑数据库的安全性问题，隔离性、一致性、合法性问题，尤其是在系统中，权限的设置尤为重要，例如在机票订票系统概念模型中，管理员的权限和用户的权限一定是被严格制定的，这样才能保证用户的隐私性。为了数据库安全性，也可以在存储数据库时进行加密，设计密文存储数据库，即密态云盘等。

2.13.5 工程师责任及其分析

工程师不仅需要完成技术上的需求，也需要承担社会责任。不仅需要完成设计数据库结构，设计相应系统体系，设计概念模型等计算机操作方面的工作，更需要考虑和分析设计的应用场景、法律效力、社会安全、隐私信息等的问题，在设计过程中，要使设计切合需求并要考虑实现成本，也要注意其法律效力，不能违背法律规定，需要在设计数据库时，同时考虑社会、健康、安全、法律以及文化等各因素之间的相互影响，让数据库更好服务于社会。

2.14 数据库应用开发(JAVA 篇)

此部分主要考察 JDBC 的体系结构、JDBC 的核心组件、使用步骤等。具体来说就是通过 JAVA 高级语言进行数据库应用系统的开发和使用，能够同时利用数据库的数据和高级语言面向对象进行编程的特性，在此部分需要通过 JDBC 实现操纵数据库的一系列操作，实现 Java 应用程序与各种不同数据库之间进行对话。此部分实验中通过了 1~7 关所有的关卡。

2.14.1 JDBC 体系结构和简单的查询

JDBC（Java DataBase Connectivity,java 数据库连接）是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。

JDBC 的核心组件包括：

（1） DriverManager： 此类管理数据库驱动程序列表。使用通信子协议将来自 java 应用程序的连接请求与适当的数据库驱动程序匹配。

（2） Driver:此接口处理与数据库服务器的通信，我们很少会直接与 Driver

对象进行交互。而是使用 DriverManager 对象来管理这种类型的对象。

（3） Connection：该界面具有用于联系数据库的所有方法。连接对象表示通信上下文，即，与数据库的所有通信仅通过连接对象。

（4） Statement：使用从此接口创建的对象将 SQL 语句提交到数据库。除了执行存储过程之外，一些派生接口还接受参数。

（5） ResultSet：在使用 Statement 对象执行 SQL 查询后，这些对象保存从数据库检索的数据。它作为一个迭代器，允许我们遍历其数据。

（6） SQLException：此类处理数据库应用程序中发生的任何错误。

构建 JDBC 应用程序涉及以下六个步骤：

（1） 导入包：需要包含包含数据库编程所需的 JDBC 类的包。大多数情况下，使用 import java.sql.*就足够了。

（2） 注册 JDBC 驱动程序：要求您初始化驱动程序，以便您可以打开与数据库的通信通道。

（3） 打开连接：需要使用 DriverManager.getConnection（）方法创建一个 Connection 对象，该对象表示与数据库的物理连接。

（4） 执行查询：需要使用类型为 Statement 的对象来构建和提交 SQL 语句到数据库。

（5） 从结果集中提取数据：需要使用相应的 ResultSet.getXXX（）方法从结果集中检索数据。

（6） 释放资源：需要明确地关闭所有数据库资源，而不依赖于 JVM 的垃圾收集。

2.14.2 用户登录

编写客户登录程序，提示用户输入邮箱和密码，并判断正确性，给出适当的提示信息。首先需要获取用户输入的用户名和密码。然后进行数据库的连接以及 JDBC 的创建，构建接口 prepareStatement 包含用户名和密码，将结果存入 resultSet 中，通过 next()方法进行查询，如果用户名和密码一致，则输出“登陆成功。”，否则输出“用户名或密码错误！”。

2.14.3 添加新客户

编程完成向 client(客户表)插入记录的方法。已知用户的 id、姓名、邮箱、

身份证号、电话和登陆密码，将上述信息与已连接的数据库的 Connection 对象传入方法中。可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现对于元组的插入操作。

具体的关键代码如下：

```
String sql = "insert into client values(?, ?, ?, ?, ?, ?)";
        try {
            PreparedStatement pps = connection.prepareStatement(sql);
            pps.setInt(1, c_id);
            pps.setString(2, c_name);
            pps.setString(3, c_mail);
            pps.setString(4, c_id_card);
            pps.setString(5, c_phone);
            pps.setString(6, c_password);
            return pps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return 0;
```

2.14.4 银行卡销户

编写一个能删除指定客户编号的指定银行卡号的方法。具体方法与插入类似。已知用户的 id 和银行卡号，将上述信息与已连接的数据库的 Connection 对象传入方法中。可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现对于元组的删除操作。

2.14.5 客户修改密码

编写修改客户登录密码的的方法。具体方法也与插入类似。首先通过一系列方法获取需要修改的用户，同时获得其旧密码，进一步对于旧密码进行更新，将用户的邮箱、旧密码和新密码与已连接的数据库的 Connection 对象传入方法中。

可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现对于元组的更新操作。

2.14.6 事务与转账操作

使用 JDBC 的事务处理编写一个银行卡转账方法。需要传入已连接数据库的 Connection 对象、转出账号、转入账号和转账金额。首先设置隔离级别为 REPEATABLE READ，具体实现为 connection. SetTransactionIsolation(4)。然后，进行转出账号扣账，转入账号还账，为储蓄卡时入账。然后进行操作合法性的检验，如果操作不合法，例如账号不存在，扣账金额不足等，则进行事务回滚，具体实现为 connection.rollback()，且置返回值为 false。其他情况下则进行事务提交，具体实现为 connection.commit()，且置返回值为 true。

2.14.7 把稀疏表格转为键值对存储

将一个稀疏的表中有保存数据的列值，以键值对(列名，列值)的形式转存到另一个表中，这样可以直接丢失没有值列。具体来说，首先需要查询稀疏表中的所有元组，进行学生学号和各科成绩的提取，然后将其填入键值对表中。

然后，将已连接的数据库的 Connection 对象和键值对传入方法中。可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现对于元组的插入操作。

3 课程总结

本次数据库系统原理实践整体完成了 2.1~2.14 的所有实训任务（除了存储过程与事务中的最后一个关卡），在整个实验过程中依次完成了数据库的基础实践，包括数据库中相关创建、删除、修改、查询等操作，进一步探索数据库的安全性控制、并发控制、自定义函数、数据库恢复等数据库中意义重大的相关技术，深入了解了数据库管理系统的整体框架和特点，对于数据库及其上层应用程序有了更加深入的理解。

对于各个实训任务，具体主要工作如下：完成了数据库、表、完整性约束条件的定义相关的所有实训任务，掌握了如何使用数据定义语言 DDL 以及如何定义约束条件；完成了表结构和完整性约束的修改相关的所有实训任务，掌握了如何利用 SQL 语句实现对表结构和约束条件的修改；完成了数据查询相关的所有实训任务，对于基本的数据查询以及数据查询过程中所用到的函数、方法、技巧有了很好的掌握；完成了数据的插入、删除、修改、连接更新相关的所有实训任务；完成了创建视图与基于视图的查询，进一步理解了模式和外模式之间的映像，掌握了应用程序是如何基于视图这一层次进行查询；完成了存储过程与事务相关的部分实训任务，掌握了三种存储过程的具体结构和实现方法；完成了触发器相关的实训任务，掌握了根据要求完成触发器的设计；完成了用户自定义函数相关的实训任务，掌握了根据要求完成自定义函数的设计；完成了创建用户、角色以及权限授予相关的所有实训任务，掌握了自主存取方式对于用户权限的设置和对于数据库的保护；完成了事务并发控制与隔离级别相关的所有实训任务，加深了对读脏、不可重复读、幻读、可串行化的理解，掌握了对于数据库事务的加锁操作；完成了使用备份和日志文件实现数据恢复相关的所有实训任务，对于数据库恢复相关技术有更加深入的理解；完成了数据库设计相关的所有实训任务，掌握了概念模型设计、关系模式设计、建模实现设计；完成了 JAVA 数据库应用开发的所有实训任务，掌握了 JDBC 体系实现数据库应用设计。

通过上述的实践过程，我获得了很大的成长，真正了解了数据库在应用程序中的重要作用，更加深入了解了如何通过数据库的系列操作实现对于数据的存储、应用，尤其是在数据查询中，我以前从未想过可以使用 SQL 语句完成如此复杂的操作，同时时间受限，没能完成 B+树相关的实验，有机会我一定要尝试一下。