# 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 |
| 班　　级： | 本硕博 2101 |
| 学　　号： | U202115666 |
| 姓　　名： | 刘文博 |
| 指导教师： | 袁平鹏 |

| | |
|---|---|
| 分数 | |
| 教师签名 | |

2023　　年　6 月　1 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

总体任务：通过头歌教学平台上的时间，引导我们掌握数据库的操作方法

“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；

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；

3. 数据库的设计与实现；
4. 数据库应用系统的开发(JAVA 篇)。

课程依托头歌实践教学平台，实践课程 url 见相关课堂教师发布链接及其邀请码。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。

# 2 任务实施过程与分析

## 2.1 数据库、表与完整性约束的定义(Create)

这一小节需要我们学会使用 MySQL 语句创建数据库，创建表以及表的主码约束，创建外码约束， CHECK， DEFAULT 和 UNIQUE 约束。

### 2.1.1 创建数据库

该关卡任务已完成，实施情况本报告略过。

### 2.1.2 创建表及表的主码约束

该关卡任务已完成，实施情况本报告略过。

### 2.1.3 创建外码约束(foreign key)

该关卡任务已完成，实施情况本报告略过。

### 2.1.4 CHECK 约束

该关卡任务已完成，实施情况本报告略过。

### 2.1.5 DEFAULT 约束

该关卡任务已完成，实施情况本报告略过。

### 2.1.6 UNIQUE 约束

该关卡任务已完成，实施情况本报告略过。

## 2.2 表结构与完整性约束的修改(ALTER)

这一小节需要我们学会使用修改表名，添加、删除和修改字段，添加、删除和修改约束。

### 2.2.1 修改表名

该关卡任务已完成，实施情况本报告略过。

### 2.2.2 添加与删除字段

该关卡任务已完成，实施情况本报告略过。

### 2.2.3 修改字段

该关卡任务已完成，实施情况本报告略过。

### 2.2.4 添加、删除与修改约束

本关主要按照要求在表中添加约束，我们可以自己给约束命名，通常用 FK/CK/UN_表名_属性名 来对外码约束 CHECK 约束 UNIQUE 约束进行命名。

## 2.3 数据查询(Select)之一

本小节主要是数据的查询。每个查询任务都需要用一条 SQL 语句完成某个查询（允许嵌套,允许使用衍生表，但只能是一条 SQL 语句，而不是两条或多条语句，除非关卡有多个任务要求）。数据集是一个事先创建并已初始化的数据库：某银行金融服务场景模拟数据库。只需写出能实现查询要求的那条 SQL 语句即可。

### 2.3.1 金融应用场景介绍,查询客户主要信息

该关卡任务已完成，实施情况本报告略过。

### 2.3.2 邮箱为 null 的客户

本关使用 is null 语句来判断邮箱是否为空，注意不能用等于 0 之类的判断语句。

### 2.3.3 既买了保险又买了基金的客户

我们可以对每个用户从 property 表中选出它购买的产品的种类，之后用一个查询选出产品种类 2，3，选出这个表中 not in 用户购买的产品的种类，如果用户既购买了保险又购买了基金那么 not in 条件会使新表为空，否则该用户就没有既买了保险又买了基金。见图 2.1

```
WHERE
    NOT EXISTS(SELECT DISTINCT property1.pro_type
    FROM property property1
    WHERE property1.pro_type <> 1 AND
    property1.pro_type NOT IN(SELECT DISTINCT property2.pro_type
    FROM
      property property2,
        client client2
    WHERE
      property2.pro_c_id=client2.c_id AND
      client2.c_id=client.c_id
        ))  # 2 3 减去 选出的用户购买的产品 为空 说明购买了基金和保险
```

图 2.1 选出题目要求的用户的 where 块

### 2.3.4 办理了储蓄卡的客户信息

本关只需要从 client 和 bank_card 表中根据条件选出相应的元素即可

### 2.3.5 每份金额在 30000～50000 之间的理财产品

本关使用 BETWEEN 30000 AND 50000 语句选出相应的理财产品

### 2.3.6 商品收益的众数

SQL 语句中的 HAVING 块在 GROUP BY 块之后执行，我们可以用 HAVING 语句对 GROUP BY 分组之后的列使用 MAX 聚集函数来得到商品收益的众数。

### 2.3.7 未购买任何理财产品的武汉居民

根据提示我们可以用 LIKE ‘4201%’来匹配武汉居民的身份证号，用 NOT EXIST 来判断筛选该客户购买的理财产品的子查询是否为空，这样就能够选出未购买任何理财产品的武汉居民。

### 2.3.8 持有两张信用卡的用户

我的做法是用一个派生表 client2 来得到持有两张信用卡的 id 属性，具体的做法是选出 COUNT(bank_card.b_c_id)>=2 的 client.id 作为 client2 表，之后从 client 选出相对应的客户的姓名，身份证号和电话号码。见图 2.2

```
SELECT
    client.c_name,
    client.c_id_card,
    client.c_phone
FROM
    client,
    (
        SELECT
            COUNT(bank_card.b_c_id) AS times,
            bank_card.b_c_id
        FROM
            bank_card
        WHERE
            bank_card.b_type = '信用卡'
        GROUP BY
            bank_card.b_c_id
        HAVING
            times
 >= 2
    ) AS client2
WHERE
    client.c_id = client2.b_c_id
ORDER BY
    client.c_id
```

图 2.2　查询语句

### 2.3.9 购买了货币型基金的客户信息

本关和上一关十分类似，我在 WHERE 块中用一个子查询得到购买了货币型基金的客户的 id，然后根据子查询的结果，选出相应客户的名称、电话号、邮箱。

### 2.3.10 投资总收益前三名的客户

本关只需要在派生表中使用一条 SUM(property.pro_income) AS total_income 语句计算投资收益，之后根据使用 tmp.total_income DESC 语句将查询结果进行降序排列，再用 LIMIT 3 语句限制记录的条数，就能够选出投资总收益前三名的客户。

### 2.3.11 黄姓客户持卡数量

该关卡任务已完成，实施情况本报告略过。本关我将 client 表和 bank_card 进行 LEFT JOIN，然后就可以使用聚集函数 COUNT 来计算持卡数量的同时选出 client 的 id 和 name 属性了。

同样用 LIKE ‘黄%’来匹配黄姓客户。

题目要求了排序的条件：按办理银行卡数量降序输出,持卡数量相同的,依客户编号排序。

用图 2.3 所示的 ORDER BY 块来实现先后的两个排序要求。

```
22  ORDER BY
23      number_of_cards DESC,
24      client.c_id ASC
```

图 2.3　ORDER BY 块

### 2.3.12 客户理财、保险与基金投资总额

本关较为复杂，但通过讲总额的计算拆解为多个子查询可以渐渐理清思路。

我的做法是让 property 表分别与 finances_product、insurance、fund 进行 INNER JOIN 计算出对应产品的投资金额，然后再将这三个子查询与 client 进行左连接，计算出客户的投资总金额。见图 2.4

```sql
SELECT
  client.c_name,
  client.c_id_card,
  COALESCE(finances.finances_amount, 0)+COALESCE(insure.insurance_amount, 0)+COALESCE(fundation.fund_amount,0) AS total_amount
FROM
  client
  LEFT JOIN
  (
    SELECT
      property.pro_c_id,
      SUM(property.pro_quantity*finances_product.p_amount) AS finances_amount
    FROM
      property
      INNER JOIN
      finances_product
      ON
        property.pro_pif_id = finances_product.p_id AND
        property.pro_type = 1
    GROUP BY
      property.pro_c_id
  ) AS finances
  ON
    client.c_id = finances.pro_c_id
  LEFT JOIN
  ( ...
  ON
    client.c_id = insure.pro_c_id
  LEFT JOIN
  ( ...
  ON
    client.c_id = fundation.pro_c_id
GROUP BY
  client.c_id,
  finances.pro_c_id,
  insure.pro_c_id,
  fundation.pro_c_id
ORDER BY
  total_amount DESC,
  client.c_id ASC
```

自动完成代码就绪. (最后更新: 立即更新

图 2.4 计算投资总金额的查询语句

### 2.3.13 客户总资产

该任务关卡跳过。

### 2.3.14 第 N 高问题

本关相对来说较为简单，理清题意之后我用如图 2.5 的子查询选出来第 4 高的保险金额。

```sql
(
    SELECT DISTINCT
        insurance.i_amount
    FROM
        insurance
    ORDER BY
        insurance.i_amount DESC
    LIMIT 3, 1
) AS tmp
```

图 2.5 选出第 4 高保险产品的保险金额

根据题意我使用 DISTINCT 关键字进行去重，使用 ORDER BY 语句实现按金额降序排列，使用 LIMIT 3，1 来跳过前 3 条数据且只取 1 条数据来获得第 4 高的金额。

### 2.3.15 基金收益两种方式排名

本关主要掌握两种排名方式：RANK() OVER() 、DENSE_RANK() OVER() 根据名字很好理解，前者是稀疏的，可以不连续，后者是稠密的，只能连续。所以前者允许同排名，后者不允许同排名。

我的做法是用一个子查询计算基金收益作为派生表和 client 表 INNER JOIN，之后就可以使用 ORDER BY 语句实现收益降序，编号升序的排列，同时在 SElECT 选出的列中增加一个计算 RANK() 得到的列作为“rank”，一个派生表计算的收益的列。

### 2.3.16 持有完全相同基金组合的客户

该任务关卡跳过。

### 2.3.17 购买基金的高峰期

该任务关卡跳过。

### 2.3.18 至少有一张信用卡余额超过 5000 元的客户信用卡总余额

本关较为简单，用 EXISTS 判断查询某一客户持有的余额超过 5000 信用卡的结果是否为空，用 SUM()聚集函数计算信用卡总余额就可以了。

### 2.3.19 以日历表格式显示每日基金购买总金额

该任务关卡跳过。

## 2.4 数据查询(Select)之二

本小节主要是数据的查询。每个查询任务都需要用一条 SQL 语句完成某个查询（允许嵌套,允许使用衍生表，但只能是一条 SQL 语句，而不是两条或多条语句，除非关卡有多个任务要求）。数据集是一个事先创建并已初始化的数据库：某银行金融服务场景模拟数据库。只需写出能实现查询要求的那条 SQL 语句即可。

### 2.4.1 查询销售总额前三的理财产品

要想查询销售总额前三的理财产品，需要计算各种理财产品的销售总额并进行排序。

而要想计算理财产品的销售总额，需要将 financial product 表和 property 表按照 finances_product.p_id = property.pro_pif_id 的条件连接，这样才可以计算 SUM(property.pro_quantity*finances_product.p_amount)也就是销售总额。

但实操过程中，如何按照题意给出 pyear 列是一个难题。通过搜集资料得知可以使用 YEAR 函数来得到日期中的 year 部分。

本关任务不难，主要掌握 rank() over(partition by ...order by)和 year（date）的用法即可。

### 2.4.2 投资积极且偏好理财类产品的客户

该任务关卡跳过。

### 2.4.3 查询购买了所有畅销理财产品的客户

定义持有人数超过 2 的理财产品称为畅销理财产品。查询购买了所有畅销理财产品的客户编号(pro_c_id)。

首先需要将 financial product 表和 property 表按照 finances_product.p_id = property.pro_pif_id 的条件连接，这样才可以根据 p_id 分组统计 COUNT(property.pro_c_id) AS num_of_client。注意只需选出 num_of_client>2 且种类为理财产品的 p_id 即可。

在选出所有畅销理财产品后，做一个子查询选出畅销理财产品表里不在某一客户购买的理财产品（同样需要一个子查询）之列的部分，如果这部分为空，那么客户就购买了所有的畅销理财产品。

在实操过程中，出现了所有客户都被选出来的情况，

```
SELECT DISTINCT
  client.c_id
FROM
  ( ... )
  client
WHERE
  NOT EXISTS (
  SELECT
    tmp.id
  WHERE
    tmp.id NOT IN ( SELECT property1.pro_pif_id FROM property property1 WHERE property1.pro_type = 1 AND property1.pro_c_id = client.c_id )
  )
ORDER BY
  client.c_id ASC
```

图 2.6 错误代码

```sql
SELECT DISTINCT
  client.c_id AS pro_c_id
FROM
  client
WHERE
  NOT EXISTS (
  SELECT
    finances_product.p_id,
    COUNT( property.pro_c_id ) AS num_of_client
  FROM
    finances_product
    INNER JOIN property ON finances_product.p_id = property.pro_pif_id
  WHERE
    property.pro_type = 1
    AND finances_product.p_id NOT IN ( SELECT property1.pro_pif_id FROM property property1 WHERE property1.pro_type = 1 AND property1.pro_c_id = client.c_id )
  GROUP BY
    finances_product.p_id
  HAVING
    num_of_client > 2
  )
ORDER BY
  client.c_id ASC
```

图 2.7 正确代码

将代码由图 2.1 修改为图 2.2，即将选出所有畅销理财产品的子查询从 from 语句移到 where 语句中，评测通过。

和助教讨论之后，没有发现什么逻辑问题，只能以后尽量把子查询都写在 where 块中。

### 2.4.4 查找相似的理财产品

该任务关卡跳过。

### 2.4.5 查询任意两个客户的相同理财产品数

该任务关卡跳过。

### 2.4.6 查找相似的理财客户

该任务关卡跳过。

## 2.5 数据的插入、修改与删除(Insert,Update,Delete)

本小节需要我们学会使用 MySQL 编写语句完成数据的插入，修改和删除。

### 2.5.1 插入多条完整的客户信息

该关卡任务已完成，实施情况本报告略过。

### 2.5.2 插入不完整的客户信息

该关卡任务已完成，实施情况本报告略过。

### 2.5.3 批量插入数据

该关卡任务已完成，实施情况本报告略过。

### 2.5.4 删除没有银行卡的客户信息

该关卡任务已完成，实施情况本报告略过。

### 2.5.5 冻结客户资产

本关需要先根据电话号码在 client 表中找到相应的客户的 c_id 再到 property 表中用 update 语句 set pro_status = “冻结”即可。

### 2.5.6 连接更新

本关需要先将 property 表和 client 表按客户 id 连接，然后 set property.pro_id_card = client.c_id_card 即可。

## 2.6 视图

本小节需要我们学会使用 MySQL 实现视图的创建和使用。

### 2.6.1 创建所有保险资产的详细记录视图

该关卡任务已完成，实施情况本报告略过。

### 2.6.2 基于视图的查询

本关只需掌握基本的聚集函数用法，并从第一关创建的视图中选择即可。

## 2.7 存储过程和事务

由于存储过程、自定义函数以及触发器，都是可编程对象，都会用到程序的控制结构和各类语句，本小节需要我们学会使用包括：变量定义语句，BEGIN...END 语句，IF 语句，CASE 语句，REPEAT 语句，WHILE 语句，游标(CURSOR)的定义和使用，事务的定义等。

### 2.7.1 使用流程控制语句的存储过程

本关使用了 while 语句和 if else 语句，掌握其用法后，只需要先声明三个变量存储斐波那契数列的计算值，一个变量记录当前的斐波那契数列的项数即可。

Sql 语句使用 set 赋值，可以使用 set tmp=a; set a=b; set b=a+tmp;语句实现斐波那契数列的迭代，同时每次循环都插入一条记录。

### 2.7.2 使用游标的存储过程

本关需要了解游标的用法，包括游标的定义，open，fetch，close 等。

首先定义两个游标，一个从所有的医生（包括主任）中遍历，一个从所有护士中遍历。其中数据集应当已经按照编号排好序。

对于护士的存储过程相对简单，只需每个循环 fetch 两次即可，得到的值作为数据项插入值班表即可。

对于医生的选择，需要考虑调班，相对比较复杂。可以用一个 doctor_tmp 存储需要调班的主任的 name，然后用一标志 exchange 存储是否需要调班。

当 dayofweek 为周一，先检查 exchange 标志，如果需要调班那么不用 fetch，直接将保存在 doctor_tmp 中的主任的 name 插入即可，同时 exchange 置 0。

当 dayofweek 为周六或周日，先 fetch 一次，然后判断取到的是不是主任的数据项，如果是那么就需要调班，将 exchange 标志置 1，主任的 name 存入 tmp，再 fetch 一次即可（因为科室只有一个主任）。

其他情况只需 fetch 一次，将取到的值插入值班表即可。

在实操过程中，由于用到两个游标还需要区分两个游标分别 not found 时的处理程序，经过请教助教，重新设计了扫描到表末尾时的判断。

格式为

```
fetch cursor_1 into nurse1_name;
if done then
close cursor_1;
open cursor_1;
set done = false;
fetch cursor_1 into nurse1_name;
end if;
```

图 2.8　扫描到表末尾时的判断

对错误代码进行修改：只设置一个 handler 处理 not found 的情况 ，增加一个 done 标志，将原本的 fetch 操作全部替换为上述 fetch 加判断的形式，先遍历 nurse 向值班表中插入一列，把里面的 nurse 的值填好，再遍历 doctor（此时的操作为 update），这样每次只会出现一个 not found 的情况。

在实验过程中还出现了语法错误，mysql 中 if else 结构如果想使用“else if”语句，要使用关键字 elseif，如果不小心出错可能很难定位这个错误。

另外，如果先用一个 while 循环对护士进行排班，然后 insert 一条不包含 n_doctor_name 的记录，然后再来一个 while 循环对医生排班，update 之前插入的记录，从逻辑上是没问题的，但是在实现中会抛出 n_doctor_name doesn`t have a default value 的错误，原因可能在于表 night_shift_schedule(夜班值班安排表)的 n_doctor_name 在建表时设定了不能为空，且没有给它设置一个默认的初始值。

将两个 while 循环合并成一个，insert 一条完整的记录才通过了本关。

### 2.7.3 使用事务的存储过程

本关较简单，只需要将操作封装为一个事务，当不满足条件时回滚事务即可。不满足的条件在题目描述中已经列出。

同时注意对于信用卡来说，b_balance 的值为欠款数目，转账过程中和储蓄卡的计算方式有区别

```
delimiter $$
create procedure sp_transfer(
    IN applicant_id int,
    IN source_card_id char(30),
    IN receiver_id int,
    IN dest_card_id char(30),
    IN amount numeric(10,2),
    OUT return_code int)
BEGIN
    declare s_owner int;
    declare d_owner int;
    declare s_balance int;
    declare s_type char(10);
    declare d_type char(10);

    select b_type ,b_c_id ,b_balance into s_type ,s_owner ,s_balance from bank_card where b_number = source_card_id;
    select b_type ,b_c_id into d_type ,d_owner from bank_card where b_number = dest_card_id;

    start transaction;

    update bank_card
    set b_balance = b_balance - amount where b_number = source_card_id ;

    if d_type = "储蓄卡" then
        update bank_card
        set b_balance = b_balance + amount where b_number = dest_card_id ;
    else
        update bank_card
        set b_balance = b_balance - amount where b_number = dest_card_id ;
    end if;

    set return_code = 1;

    if s_owner <> applicant_id or d_owner <> receiver_id or s_balance < amount or (s_type = "信用卡" and d_type = "储蓄卡") then
      begin
        rollback ;
        set return_code = 0;
      end ;
    else
      commit ;
    end if;
END$$
delimiter ;
```

图 2.9 使用事务的存储过程的代码

## 2.8 触发器

触发器(trigger)是对约束(constraints)的补充，它用程序来完成 constraint 实现不了的业务规则。如果在表上定义了 insert,delete 或 update 事件驱动的触发器，那么当执行 insert,delete 或 update 语句时，会激活相应的触发器，以检查数据的完整性。本实训的任务是用 create trigger 语句，创建符合任务要求的触发器

### 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

本关需要掌握触发器的用法。首先创建触发器，之后用 if then 语句设置触发条件，使用 concat 函数构造错误信息，再由 SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg 语句抛出错误即可。触发条件和应构造的错误信息在题目描述中已经给出。

在实操时出现了语法错误(见附录图 2.10、2.11)，由于 SQL 报错信息定位不准，我选择询问助教，之后选择重写代码(见附录图 2.12)，终于顺利通过。

但过了很久，我发现我误将 elseif 写成了 else if。

## 2.9 用户自定义函数

本小节需要我们创建一个完成指定任务的函数，并在语句中使用它

### 2.9.1 创建函数并在语句中使用它

本关内容相对简单，只需要类似存储结构地定义题目要求的函数，之后这个函数就可以像系统里的聚集函数那样直接使用了。这样可以直接用一条 sql 语句实现复杂的功能，更加简洁方便。

## 2.10 安全性控制

安全性一般要从全方位、多层次考虑具体的安全措施，比如不仅要考虑 MySQL 的安全性，还要考虑服务器主机的安全性(如防 eavesdropping, altering, playback, 以及 DoS 攻击等)。本小节需要完成 MySQL 安全性中的一部分：存取控制。

### 2.10.1 用户与权限

按照要求写出正确的 grant，revoke 语句即可。创建用户时需注意用户名和域名分别要用 ’’ 隔开，中间加上@才是正确的。

```
# 请填写语句，完成以下功能：
#(1) 创建用户tom和jerry，初始密码均为'123456'；
create user 'tom','jerry' identified by '123456';
#(2) 授予用户tom查询客户的姓名，邮箱和电话的权限,且tom可转授权限；
grant select (c_name,c_mail,c_phone) on client to tom with grant option;
#(3) 授予用户jerry修改银行卡余额的权限；
grant update (b_balance) on bank_card to jerry ;
#(4) 收回用户Cindy查询银行卡信息的权限。
revoke select on bank_card from Cindy;
```

图 2.13 创建用户、授予用户权限的 SQL 语句

### 2.10.2 用户、角色与权限

本关使用了 role 的相关语句，按照要求给出正确语句即可。

## 2.11 并发控制与事务的隔离级别

多个事务并发访问数据，如果不加以控制，极易导致数据的不一致性问题。本小节先介绍并发控制和事务的隔离级别，接着要求通过设置恰当的的隔离级别，

编写两个事务构造读脏、不可重复读、幻读等场景，最后要求在 read uncommitted 隔离级别下，手工加锁，保证不可重复读。

### 2.11.1 并发控制与事务的隔离级别

该关卡任务已完成，实施情况本报告略过。

### 2.11.2 读脏

根据隔离等级的相关知识，只有最低级的 read uncommited 能够容忍读脏。同时使用等待语句，让事务二先 update，事务一后查询，同时确保事务一查询之后，事务二才撤销更改，在事务二撤销更改之后，事务一 commit。此时，事务一读到的数据就会和数据库中的数据不一样，出现 dirty read。

```
-- 事务1:
use testdb1;
## 请设置适当的事务隔离级别
set session transaction isolation level read uncommitted;

start transaction;

-- 时刻2 - 事务1读航班余票,发生在事务2修改之后
## 添加等待代码，确保读脏
set @n = sleep(5);
select tickets from ticket where flight_no = 'CA8213';
set @n = sleep(10);
commit;

```

图 2.14 事务一代码

```
-- 事务2
use testdb1;
## 请设置适当的事务隔离级别
set session transaction isolation level read uncommitted;
start transaction;
-- 时刻1 - 事务2修改航班余票
update ticket set tickets = tickets - 1 where flight_no = 'CA8213';

-- 时刻3 - 事务2 取消本次修改
## 请添加代码，使事务1在事务2撤销前读脏;
set @n = sleep(5);
rollback;

```

图 2.15 事务二代码

### 2.11.3 不可重复读

| time_seq | t | tickets |
|---|---|---|
| 1 | 159 | |
| 2 | 159 | |
| 3 | 158 | |
| 4 | 158 | |
| 5 | 157 | |
| 6 | 157 | |

图 2.16　不可重复读预期输出

根据预期输出和代码框架，推测两个事务的执行过程。易知需要通过等待语句实现，事务一先读取航班余票，事务二后读取航班余票，事务二读取后事务一修改航班余票并读取，之后事务二读取航班余票之后再次修改，之后事务一commit，之后事务二 commit。

通过合理设计等待时间，避免死锁和时刻顺序出错即可通过本关。

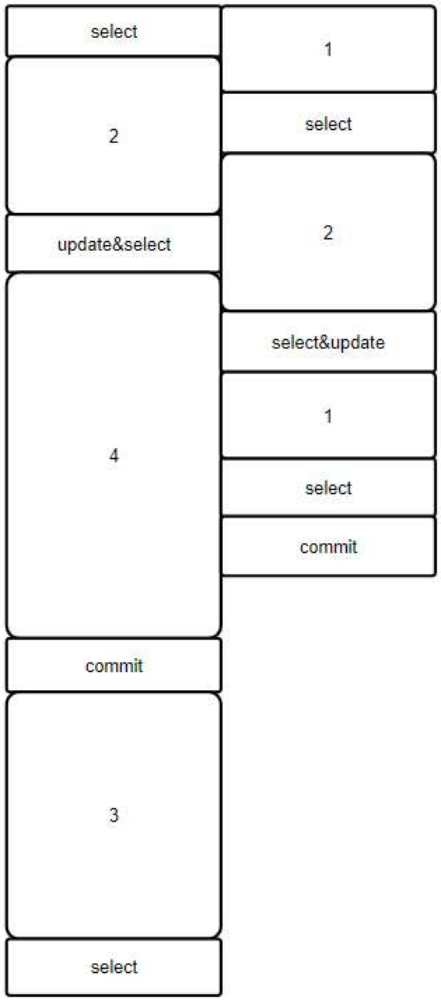



图 2.17　根据题意设计相应的等待语句

值得注意的是，如果事务二的隔离等级为 read committed，即使时刻顺序正确，事务二在事务一修改航班余票但并没有提交时读到的数据不会被修改，但当事务二修改了航班余票后即使没有两个事务都没有提交，事务二仍然可以读到共同修改后的数据。

见附录中的图 2.18-2.22。

### 2.11.4 幻读

如果两次查询之间没有 sleep 语句，由于隔离等级为可重复读，那么两次查询的结果会相同（原因在于可重复读会锁定查询的行，保存查询结果，防止其被 update 从而产生不可重复读）。

```
1   -- 事务1（采用默认的事务隔离级别- repeatable read）：
2   use testdb1;
3   select @@transaction_isolation;
4   start transaction;
5   ## 第1次查询余票超过300张的航班信息
6   select * from ticket where tickets > 300;
7
8   -- 修改航班MU5111的执飞机型为A330-300：
9   update ticket set aircraft = 'A330-300' where flight_no = 'MU5111';
10  -- 第2次查询余票超过300张的航班信息
11  select * from ticket where tickets > 300;
12  commit;
```

图 2.23　两次查询之间没有等待语句


图 2.24　评测结果

如果加上等待语句，那么虽然原来的行仍然被保存，但也会读到新修改的行，出现幻读的现象。（可能是因为该隔离等级下给查询结果所在行加的锁是非阻塞锁）。

### 2.11.5 主动加锁保证可重复读

根据题目要求，事务一要实现可重复读，那么需要在第一次查询时给 ticket 表加上读锁，使得事务二此时不能对 ticket 表进行写操作，保证两次查询的一致性。之后事务一 commit 后，查询得到修改后的值。

### 2.11.6 可串行化

默认的隔离级别是 repeatable read，在此基础上实现可串行化。若想让并行的结果和先进行事务二再进行事务一保持一致，我们可以给事务二加上一个锁，由于事务二执行 update 操作，选择给事务二加上 write 锁。在事务一一开始加上一个等待语句即可。

```
1  -- 事务1:
2  use testdb1;
3  start transaction;
4  set @n = sleep(1);
5  select tickets from ticket where flight_no = 'MU2455';
6  select tickets from ticket where flight_no = 'MU2455';
7  commit;
8
```

图 2.25　事务一代码

```
1  -- 事务2:
2  use testdb1;
3  start transaction;
4  select NULL from ticket where 1=0 for update;
5  update ticket set tickets = tickets - 1 where flight_no = 'MU2455' ;
6  commit;
7
```

图 2.26　事务二代码（无意义的 select 的语句起加锁的作用）

## 2.12 备份+日志：介质故障与数据库恢复

MySQL 在事务机制的保障下，利用备份、日志文件实现恢复。在生产环境中，通常需要部署多台服务器，可以利用复制、镜像、master-slave 等机制保障数据的安全性、可用性，同时提供高并发服务。 这一节需要我们学会如何备份

### 2.12.1 备份与恢复

本关只需要根据 mysql 和 mysqldump 的相关知识，写出对应的备份和恢复指令即可。

### 2.12.2 备份+日志：介质故障的发生与数据库的恢复

该任务关卡跳过。

## 2.13 数据库设计与实现

数据库设计的主要过程可以概括为：在精准需求分析的基础上，为数据库建模，从概念模型到逻辑模型，再到物理模型，即在某个具体的 DBMS 上实现，最后是应用系统的实施和运维。其中最重要的工作是数据库建模。 本小节需要我们完成概念模型、逻辑模型和物理模型的设计。

### 2.13.1 从概念模型到 MySQL 实现

根据题目描述和 gitee 上的 Crow's foot ER 图易知具体数据库该如何设计。

### 2.13.2 从需求分析到逻辑模型

该任务关卡跳过。

### 2.13.3 建模工具的使用

该任务关卡跳过。

### 2.13.4 制约因素分析与设计

数据库作为一个信息的集合体，管理者在实际使用过程中必须注意对用户权限的管理与限制，否则稍有不慎就可能造成重要数据的泄露甚至恶意使用。这种现象在当今的大数据时代尤为常见，例如超星学习通等软件的信息泄露，所造成的影响是很大的。

同时数据库的管理者还应该做好防范恶意攻击和出现故障进行修复，保障用户的信息安全。

数据库应用于生产管理方面，规模化的批量工作代替传统的人工操作模式，大大提升了工作效益和工作质量，方便了人们的生活，也不断地推进着社会的生产力的发展；因此说数据库技术在我们现代社会中起着不可忽视的作用，数据技术也正推动着时代的进步。 但我们同样需要意识到法律，文化，安全等这些稍有差错就有可能酿成大错的因素，这就需要我们在设计过程中充分考虑各种可能性，尽可能交付一个足够完美的作品。

### 2.13.5 工程师责任及其分析

作为一名工程师，需要能够基于工程相关背景知识进行合理分析，评价专业工程实践和复杂工程问题解决方案对社会、健康、安全、法律以及文化的影响，并理解应承担的责任。同时还要能够能够基于科学原理并采用科学方法对复杂工程问题进行研究，包括设计实验、分析与解释数据、并通过信息综合得出合理有效的结论。并且还要能够理解和评价针对复杂工程问题的工程实践对环境、社会可持续发展的影响。

## 2.14 数据库应用开发(JAVA 篇)

数据库应用开发，系指利用高级语言开发数据库应用系统，这些应用系统可是 C/S 架构的，也可以是 B/S 架构的(即 web 应用)，开发的语言可是 C++,JAVA， PHP， JSP， Python 等。 本小节通过几个简单的练习来掌握 JAVA 开发数据库应用的基本知识。

### 2.14.1 JDBC 体系结构和简单的查询

本关需要熟悉 JDBC 的使用步骤，包括导入 JDBC 包 import java.sql.*;

注册 JDBC 驱动程序 Class.forName("com.mysql.cj.jdbc.Driver");

数据库 URL 配置，本关用到 finance 数据库

```
String URL = "jdbc:mysql://127.0.0.1:3306/finance?useUnicode=true&characterEncoding=UTF8&useSSL=false&serverTimezone
String USER = "root";
String PASS = "123123";
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

图 2.27 数据库 URL 配置部分代码

先创建数据库连接对象

再按需求执行 stetement 或 preparedstatement 对象

```
Connection conn = DriverManager.getConnection(URL, USER, PASS);
String SQL = "SELECT client.c_name AS `姓名`, client.c_mail AS `邮箱`, client.c_phone AS `电话` FROM client WHERE client.c_mail IS NOT NULL
statement = conn.createStatement();
resultSet = statement.executeQuery(SQL);
```

图 2.28 创建和执行数据库对象

最终得到结果。

Java 采用 try • catch • finally 的结构，用来执行 • 发现并处理异常 • 清理打开的资源。

### 2.14.2 用户登录

本关主要是掌握条件不确定的查询，因为关卡中的 loginName 和 loginPass 在输入后才知道是什么，所以要在 SQL 语句中用某种方式代替它。

我是用的方法是直接拼接 String SQL="SELECT c_id FROM client WHERE c_mail = '"+loginName+"' AND c_password= '"+loginPass+"' ;"; 当然也可以用 preparedStatement 语句来实现。

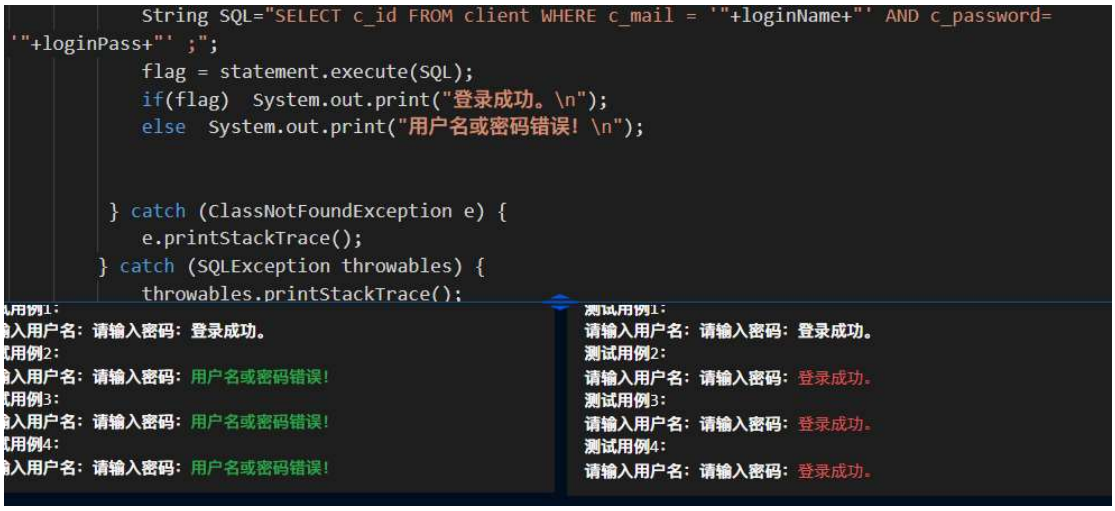



图 2.29 错误代码示例

值得一提的是，我在完成关卡时一开始以为 statement.execute（SQL）的返回值是根据 resultSet 是否为空决定的，我认为如果返回值为 false 就说明匹配不到，true 就说明输入的 c_mail 和 c_password 是正确的。

但实际上 statement.execute（SQL）的返回值是根据表是否存在来返回的，即使表是空的也会返回 true。

将其用 statement.execute(SQL)替换并用 if(resultSet.next()) 来判断就能轻松通过本关了。

### 2.14.3 添加新客户

本关主要考察 PreparedStatement 和 executeUpdate()的使用。

注意 c_id 是 int 类型，但填充到 SQL 语句时要用 String.valurOf()函数来转换为 string 类型。

如果只修改对象内部的内容会报错。

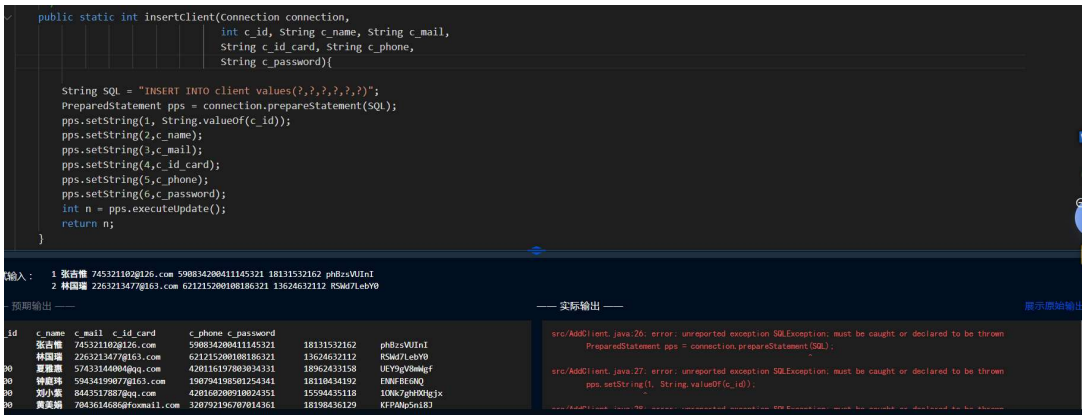



图 2.30 出现异常

在对象的声明后面加上 throws SQLExcept 则可以通过本关。

### 2.14.4 银行卡销户

该关卡任务已完成，实施情况本报告略过。

### 2.14.5 客户修改密码

本关的重点时判断不同的情形给出返回值。

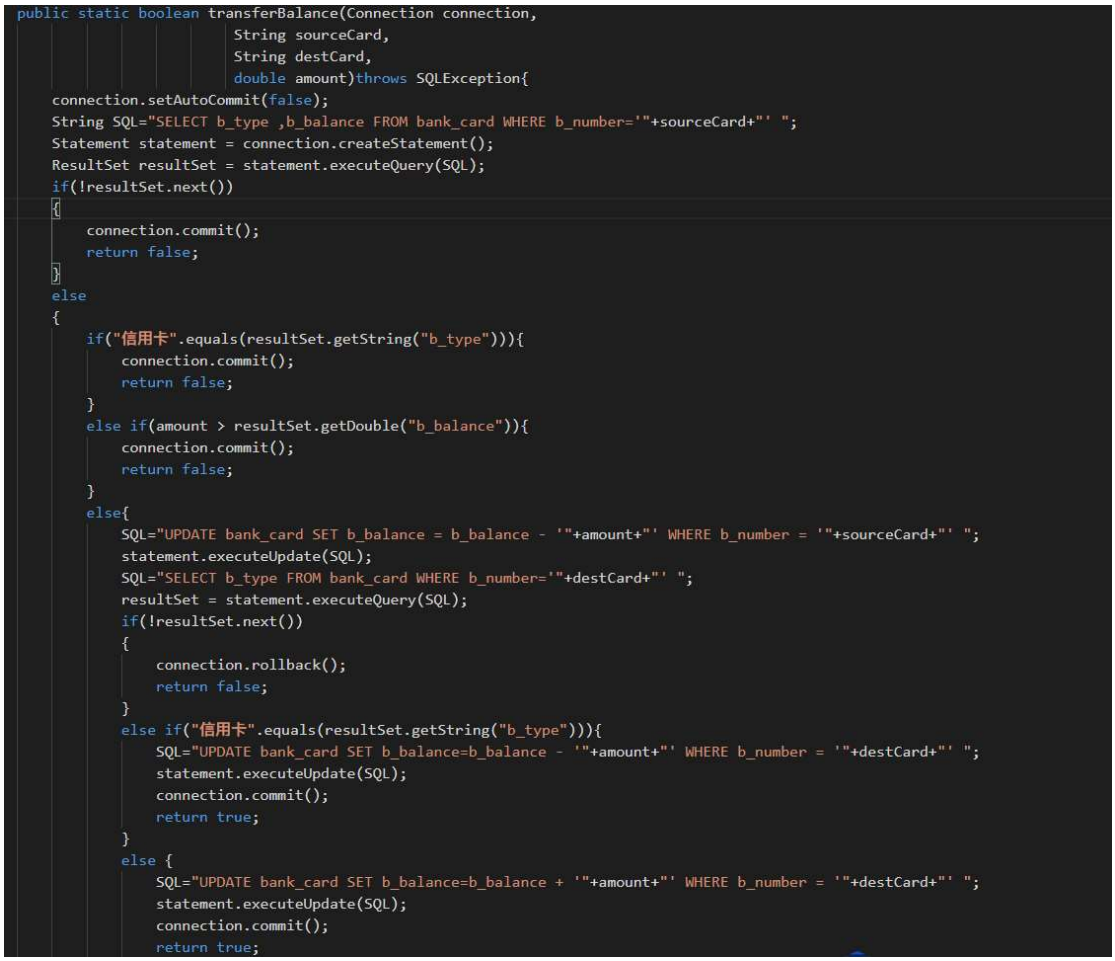
具体来说，passwd()方法返回一个整数： 1 - 密码修改成功 2 - 用户不存在 3 - 密码不正确 -1 - 程序异常(如没能连接到数据库等）

在执行 update 语句之前对情况 2，3 进行判断，添加一个 try catch 结构，用 catch（Exception e）语句来捕获所有程序异常，即可通过本关。

### 2.14.6 事务与转账操作

本关主要考察 JDBC 的事务处理，我们先设置 connection.setAutoCommit(false);这样 connection 不会自动提交。

修改后的 transferBalance（）对象见下图。



```
public static boolean transferBalance(Connection connection,
                        String sourceCard,
                        String destCard,
                        double amount)throws SQLException{
    connection.setAutoCommit(false);
    String SQL="SELECT b_type ,b_balance FROM bank_card WHERE b_number='"+sourceCard+"' ";
    Statement statement = connection.createStatement();
    ResultSet resultSet = statement.executeQuery(SQL);
    if(!resultSet.next())
    {
        connection.commit();
        return false;
    }
    else
    {
        if("信用卡".equals(resultSet.getString("b_type"))){
            connection.commit();
            return false;
        }
        else if(amount > resultSet.getDouble("b_balance")){
            connection.commit();
            return false;
        }
        else{
            SQL="UPDATE bank_card SET b_balance = b_balance - '"+amount+"' WHERE b_number = '"+sourceCard+"' ";
            statement.executeUpdate(SQL);
            SQL="SELECT b_type FROM bank_card WHERE b_number='"+destCard+"' ";
            resultSet = statement.executeQuery(SQL);
            if(!resultSet.next())
            {
                connection.rollback();
                return false;
            }
            else if("信用卡".equals(resultSet.getString("b_type"))){
                SQL="UPDATE bank_card SET b_balance=b_balance - '"+amount+"' WHERE b_number = '"+destCard+"' ";
                statement.executeUpdate(SQL);
                connection.commit();
                return true;
            }
            else {
                SQL="UPDATE bank_card SET b_balance=b_balance + '"+amount+"' WHERE b_number = '"+destCard+"' ";
                statement.executeUpdate(SQL);
                connection.commit();
                return true;
```

图 2.31　transferBalance( )部分

我的思路是首先查询转出账号的类型和余额，如果查询结果为空或者查到的结果类型为“信用卡”或余额小于 amount，就 connection.commit（）返回 false。

否则，update 转出卡的余额，但此时不会 commit

之后接着查询转入账号，如果该账号不存在，connection.rollback（）然后返回 false。

否则，根据转入账号的类型，更改其余额，之后 connection.commit（），返回 true。

JDBC 下的事务处理给予了更高的自由度，允许我一步一步执行或回滚事务，这一点是十分便利的。

### 2.14.7 把稀疏表格转为键值对存储

该关主要是理解对象、方法之间的关系和使用，以及了解将稀疏表格转为键值对存储的这种思想。

根据题意，设计一个 insertSC（）方法，用于向 sc 表中插入记录。

在 main 函数根据已经给出的 url，user，password 和 driver 中建立连接，之后选出 entrance_exam 表的所有数据进行遍历，依次调用 insertSC（）方法插入到 sc 表中即可。

```
13      public static int insertSC(Connection connection,
14                                 int sno,
15                                 String col_name,
16                                 int col_value)throws SQLException{
17      if(col_value!=0){
18      String SQL = "insert into sc values(?,?,?)";
19      PreparedStatement pps = connection.prepareStatement(SQL);
20      pps.setString(1, String.valueOf(sno));
21      pps.setString(2,col_name);
22      pps.setString(3,String.valueOf(col_value));
23      pps.executeUpdate();
24      return 0;
25      }
26      else return 1;
27  }
```

图 2.32　在对象内直接用=0 的条件

注意到这里科目的成绩为空，在方法里可以直接用=0 的条件判断出来。

## 2.15 数据库的索引 B+树实现

B+树索引结点类型的数据结构设计主要包括：BPlusTreePage 的设计、BPlusTreeInternalPage 的设计、BPlusTreeLeafPage 的设计。

其功能包括：B+树索引：Insert、B+树索引：Remove

本小节的目标是了解 B+树索引节点类型的数据结构及其功能的设计。

### 2.15.1 BPlusTreePage 的设计

该任务关卡跳过。

### 2.15.2 BPlusTreeInternalPage 的设计

该任务关卡跳过。

### 2.15.3 BPlusTreeLeafPage 的设计

该任务关卡跳过。

### 2.15.4 B+树索引：Insert

该任务关卡跳过。

### 2.15.5 B+树索引：Remove

该任务关卡跳过。

# 3 课程总结

这门实验的主要任务是在头歌平台上完成相关关卡，整个实验共分为 12 个大关卡，通过每个关卡中的每个小实验可以获得分数，最终我在头歌平台上的得分是 102 分。

| | | | | |
|---|---|---|---|---|
| 刘文博 | 计算机2021级本硕博班 | 102 | 0 | 102 |

这门实验的主要工作是

1. 掌握如何创建数据库：包括建库、建表、加入约束
2. 掌握如何使用数据库：插入、删除、视图以及各式各样的查询
3. 掌握数据库的个性化使用：用户自定义的函数、安全性控制以及数据库的应用开发
4. 对数据库深入理解：数据库的设计与恢复流程、事务的并发控制、B+树索引的实现

这门实验总的来说让我了解了数据库的使用，和数据库系统原理课程结合在一块让我对数据库这个领域同时有了理论和实践经验。也让我体会到 SQL 语言的非过程化，这使得 SQL 语言在使用时可以专注于设计而不用考虑具体实现细节。我认为这门实验的设计还是相当好的，头歌平台上的题目分值超出规定的 100 分一大半，使得我在实践过程中更加自由，而部分关卡限制跳关又能让同学们不会忽略掉重点内容。通过这次实验，我已经熟悉了数据库的许多操作，希望能在未来把这项技能用到实处，也希望数据库实验课可以越办越好。

# 附录

```sql
use finance1;
drop trigger if exists before_property_inserted;
-- 请在适当的地方补充代码，完成任务要求：

delimiter $$
CREATE TRIGGER before_property_inserted BEFORE INSERT ON property
REFERENCING NEW ROW AS newturple
FOR EACH ROW
BEGIN
  declare msg char(127);
  if newturple.pro_type not in (1,2,3) then
     set msg = concat('type ', newturple.pro_type, ' is illegal! ');
     SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
  else if newturple.pro_type = 1 then
     if newturple.pro_pif_id not in (select p_id from finances_product) then
        set msg = concat('finances product #',newturple.pro_pif_id,' not found! ');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
     end if;
  else if newturple.pro_type = 2 then
     if newturple.pro_pif_id not in (select i_id from insurance) then
        set msg = concat('insurance #',newturple.pro_pif_id,' not found! ');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
     end if;
  else if newturple.pro_type = 3 then
     if newturple.pro_pif_id not in (select f_id from fund) then
        set msg = concat('fund #',newturple.pro_pif_id,' not found! ');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
     end if;
  end if;
END$$
delimiter ;

```

图 2.10 触发器错误代码

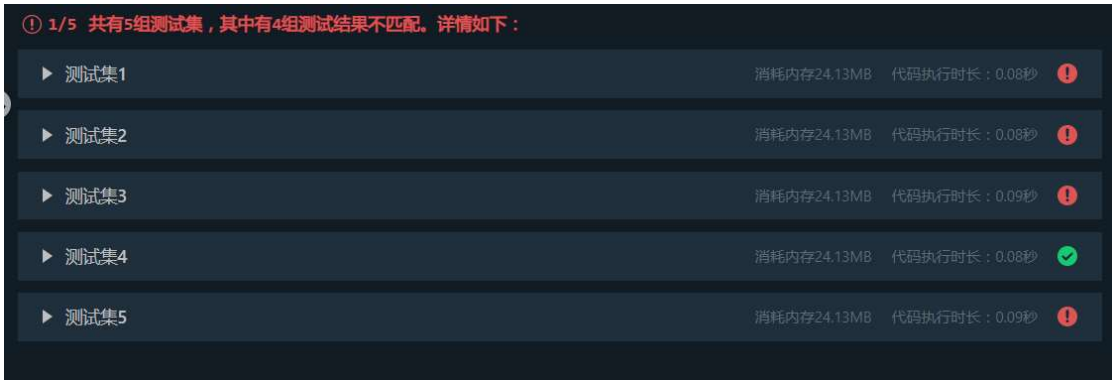



图 2.11 评测结果

```
use finance1;
drop trigger if exists before_property_inserted;
-- 请在适当的地方补充代码，完成任务要求：
delimiter $$
CREATE TRIGGER before_property_inserted BEFORE INSERT ON property
FOR EACH ROW
BEGIN
if NEW.pro_type=1 and NEW.pro_pif_id not in (select p_id from finances_product)
then set @msg = CONCAT('finances product #', NEW.pro_pif_id, ' not found!');
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
end if;

if NEW.pro_type=2 and NEW.pro_pif_id not in (select i_id from insurance)
then set @msg = CONCAT('insurance #', NEW.pro_pif_id, ' not found!');
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
end if;

if NEW.pro_type=3 and NEW.pro_pif_id not in (select f_id from fund)
then set @msg = CONCAT('fund #', NEW.pro_pif_id, ' not found!');
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
end if;

if NEW.pro_type > 3 or NEW.pro_type < 1
then set @msg = CONCAT('type ', NEW.pro_type, ' is illegal!');
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
end if;

END$$
delimiter ;
```

图 2.12 触发器正确代码

```sql
-- 事务1:
## 请设置适当的事务隔离级别
set session transaction isolation level read committed;

-- 开启事务
start transaction;
-- 时刻1 - 事务1读航班余票:
insert into result
select now(),1 t, tickets from ticket where flight_no = 'CZ5525';

## 添加等待代码，确保事务2的第一次读取在事务1修改前发生
set @n = sleep(2);

-- 时刻3 - 事务1修改余票，并立即读取:
update ticket set tickets = tickets - 1 where flight_no = 'CZ5525';
insert into result
select now(),1 t, tickets from ticket where flight_no = 'CZ5525';

## 添加代码，使事务2 的第2次读取在事务1修改之后，提交之前发生
set @n=sleep(4);
commit;

-- 时刻6 - 事务1在t2也提交后读取余票
## 添加代码，确保事务1在事务2提交后读取

set @n = sleep(3);
insert into result
select now(), 1 t, tickets from ticket where flight_no = 'CZ5525';

```

图 2.18　事务一隔离等级为 read committed

```sql
-- 事务2
## 请设置适当的事务隔离级别以构造不可重复读
set session transaction isolation level read uncommitted;
start transaction;
-- 时刻2 - 事务2在事务1读取余票之后也读取余票
## 添加代码，确保事务2的第1次读发生在事务1读之后，修改之前
set @n = sleep(1);

insert into result
select now(),2 t, tickets from ticket where flight_no = 'CZ5525';

-- 时刻4 - 事务2在事务1修改余票但未提交前再次读取余票，事务2的两次读取结果应该不同
## 添加代码，确保事务2的读取时机
set @n = sleep(2);
insert into result
select now(), 2 t, tickets from ticket where flight_no = 'CZ5525';

-- 事务2立即修改余票
update ticket set tickets = tickets - 1 where flight_no = 'CZ5525';

-- 时刻5 - 事务2 读取余票（自己修改但未交的结果）:
 set @n = sleep(1);
insert into result
select now(), 2 t, tickets from ticket where flight_no = 'CZ5525';

commit;
```

图 2.19　事务二隔离等级为 read uncommitted


图 2.20　评测结果

```
1  -- 事务2
2  ## 请设置适当的事务隔离级别以构造不可重复读
3  set session transaction isolation level read committed;
```

图 2.21　修改事务二隔离等级为 read committed

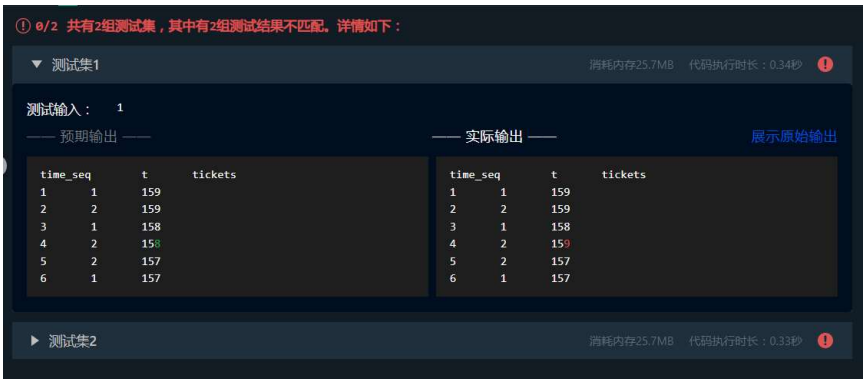



图 2.22　时刻顺序没有改变但事务二读到的数据仍为原数据