上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

数据库技术第二次上机作业

实验报告

学生姓名: 翟文祺

学生学号: 516141910032

第二次上机关系数据库编程大作业

516141910032 翟文祺

0.0.**版本说明：**
Python／anaconda： python 3
MySQL Workbench version：5.2.47
mysql.connector version：2.1.3
0.1.**文件说明：**
EmbeddedSQL.py –嵌入式 SQL 文件 （在 anaconda 中实现后 copy 代码到 idle 中）
procedure.sql－两个存储过程的实现文件
trigger.sql－两个触发器的实现文件

第一部分：在线购物数据库的构建
1.系统的可行性分析
1.1. 根据法则：表满足以下条件时就是第三范式(满足 3NF 条件)：
- 满足第二范式，即数据表中**不存在非主属性对于码的部分函数依赖**
- 所有非主字段都是依赖于主键

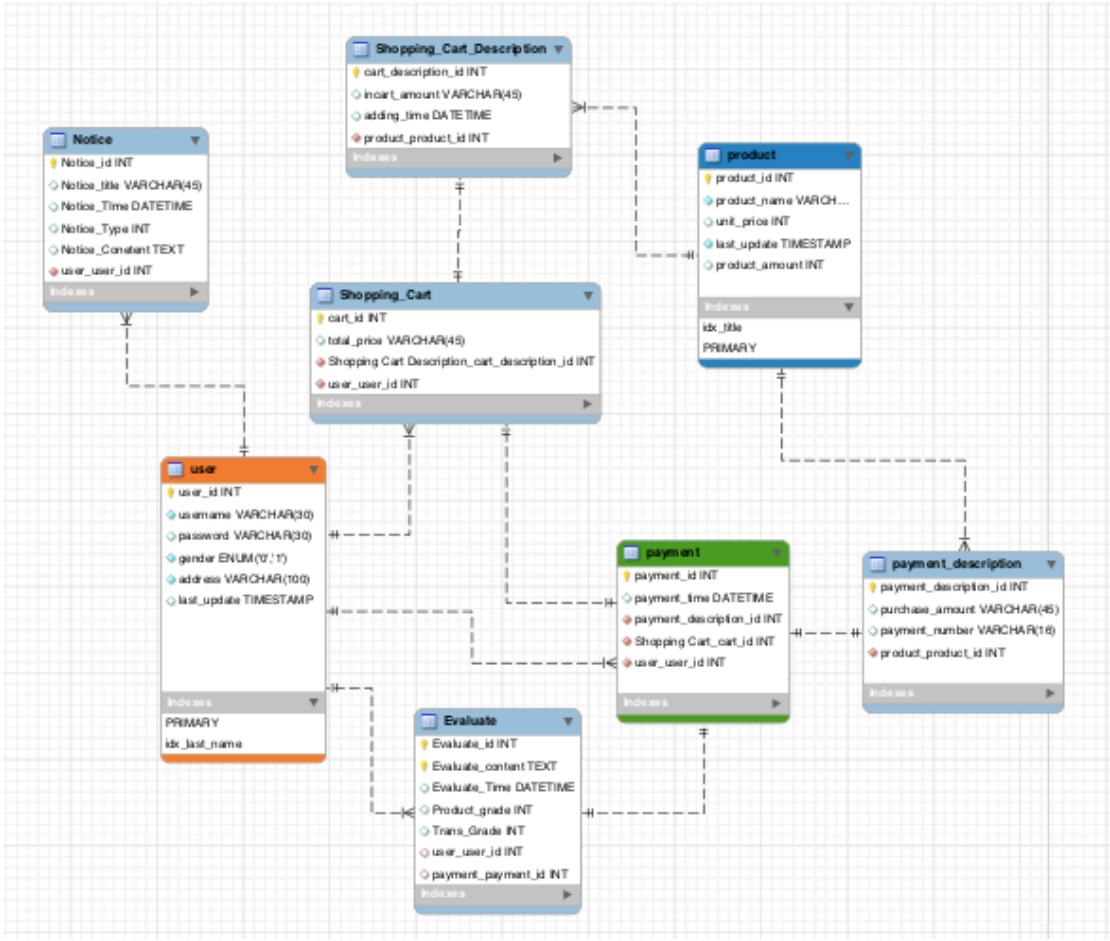
1.2. 对于题目给定关系的 3NF 分析：
原始 ppt 上给出的关系不满足 3NF 范式
1. 用户表（用户 ID，用户名,用户密码，用户性别，收货地址）
分析： 用户 id 为主键，主属性，用户名，密码，性别，收货地址同时**满足**第二范式和第三范式，因为非主字段仅依赖于主键，非主字段内部没有联系。
2. 商品表（商品名,商品价格,剩余库存）
分析： 商品 id 为主键，主属性，商品名，价格，库存同时**满足**第二范式和第三范式，因为非主字段，仅依赖于主键。
3. 订单表（订单 ID，购物车 ID，用户 ID，订单编号，订单金额，下单时间）
分析：订单 ID 为主键，购物车 ID 和用户 ID **不满足**第三范式，购物车 ID 是购物车表的主键，在订单表中应该以外键形式存在，用户 ID 同理。同时应该添加订单详情 ID 作为 foreign key 插入表格中。
4. 订单详情表（订单详情 ID，订单 ID，商品 ID，商品数量，商品价格）
分析：该表为子表，依赖于订单表存在。订单 ID 不应该出现，否则母子表顺序颠倒，订单详情 ID 为主键，为本表的主属性，作为 foreign key 存在于订单表中。该表中商品数量和商品价格不满足第三范式，因为这两个变量依赖于商品 ID，具体来看因为订单 ID →商品 ID，同时商品 ID→商品价格，所以存在非主属性对于码的传递函数依赖。（订单详情 ID，商品 ID） →商品数量，存在非主属性 数量 对码（订单详情 ID，商品 ID）的部分函数依赖，不满足 2NF，**更不满足 3NF**。
5. 购物车表（购物车 ID，用户 ID，商品总价）
分析：购物车 ID 为主键，给定购物车 ID，有购物车 ID →用户 ID，购物车 ID →商品总价，非主属性内部不存在传递函数依赖，**满足第三范式**。
6. 购物车详情表（购物车详情 ID，购物车 ID，商品 ID，商品数量，添加时间，商品价格）

分析：该表为子表，依赖于订单表存在。购物车 ID 不应该出现，否则母子表顺序颠倒，购物车详情 ID 为主键，为本表的主属性，作为 foreign key 存在于订单表中。该表中商品数量和商品价格不满足第三范式，因为这两个变量依赖于商品 ID，具体来看因为订单 ID →商品 ID，同时商品 ID→商品价格，所以存在非主属性对于码传递函数依赖，不满足 3NF。（订单详情 ID，商品 ID）→商品数量，存在非主属性 **数量** 对码（购物单详情 ID，商品 ID）的部分函数依赖，**不满足 3NF。**

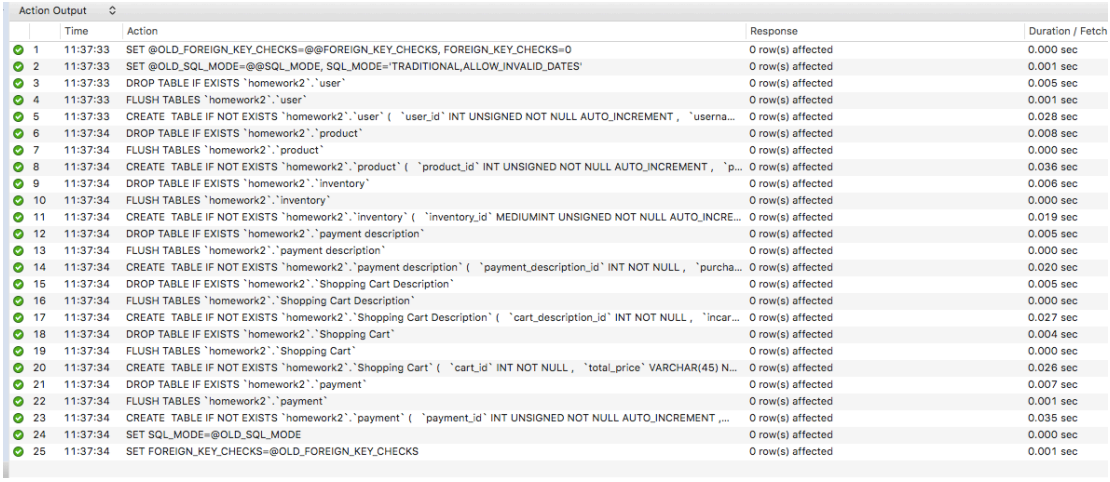
2.数据库设计－ER 模型

包括 8 个表：

用户（user）,购物车（shopping_cart）,购物车详情（shopping_cart_description）,商品（product），订单（payment），订单详情（payment_description）（附加题又添加了评价（Evaluate）和新闻公告（Notice））



通过 Forward Engineer 的方式生成 Script，构建数据库构建成功



Action Output

	Time	Action	Response	Duration / Fetch
1	11:37:33	SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0	0 row(s) affected	0.000 sec
2	11:37:33	SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES'	0 row(s) affected	0.001 sec
3	11:37:33	DROP TABLE IF EXISTS `homework2`.`user`	0 row(s) affected	0.005 sec
4	11:37:33	FLUSH TABLES `homework2`.`user`	0 row(s) affected	0.001 sec
5	11:37:33	CREATE TABLE IF NOT EXISTS `homework2`.`user` (`user_id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `userna...	0 row(s) affected	0.028 sec
6	11:37:34	DROP TABLE IF EXISTS `homework2`.`product`	0 row(s) affected	0.008 sec
7	11:37:34	FLUSH TABLES `homework2`.`product`	0 row(s) affected	0.000 sec
8	11:37:34	CREATE TABLE IF NOT EXISTS `homework2`.`product` (`product_id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `p...	0 row(s) affected	0.036 sec
9	11:37:34	DROP TABLE IF EXISTS `homework2`.`inventory`	0 row(s) affected	0.006 sec
10	11:37:34	FLUSH TABLES `homework2`.`inventory`	0 row(s) affected	0.000 sec
11	11:37:34	CREATE TABLE IF NOT EXISTS `homework2`.`inventory` (`inventory_id` MEDIUMINT UNSIGNED NOT NULL AUTO_INCRE...	0 row(s) affected	0.019 sec
12	11:37:34	DROP TABLE IF EXISTS `homework2`.`payment description`	0 row(s) affected	0.005 sec
13	11:37:34	FLUSH TABLES `homework2`.`payment description`	0 row(s) affected	0.000 sec
14	11:37:34	CREATE TABLE IF NOT EXISTS `homework2`.`payment description` (`payment_description_id` INT NOT NULL , `purcha...	0 row(s) affected	0.020 sec
15	11:37:34	DROP TABLE IF EXISTS `homework2`.`Shopping Cart Description`	0 row(s) affected	0.005 sec
16	11:37:34	FLUSH TABLES `homework2`.`Shopping Cart Description`	0 row(s) affected	0.000 sec
17	11:37:34	CREATE TABLE IF NOT EXISTS `homework2`.`Shopping Cart Description` (`cart_description_id` INT NOT NULL , `incar...	0 row(s) affected	0.027 sec
18	11:37:34	DROP TABLE IF EXISTS `homework2`.`Shopping Cart`	0 row(s) affected	0.004 sec
19	11:37:34	FLUSH TABLES `homework2`.`Shopping Cart`	0 row(s) affected	0.000 sec
20	11:37:34	CREATE TABLE IF NOT EXISTS `homework2`.`Shopping Cart` (`cart_id` INT NOT NULL , `total_price` VARCHAR(45) N...	0 row(s) affected	0.026 sec
21	11:37:34	DROP TABLE IF EXISTS `homework2`.`payment`	0 row(s) affected	0.007 sec
22	11:37:34	FLUSH TABLES `homework2`.`payment`	0 row(s) affected	0.001 sec
23	11:37:34	CREATE TABLE IF NOT EXISTS `homework2`.`payment` (`payment_id` INT UNSIGNED NOT NULL AUTO_INCREMENT ,...	0 row(s) affected	0.035 sec
24	11:37:34	SET SQL_MODE=@OLD_SQL_MODE	0 row(s) affected	0.000 sec
25	11:37:34	SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS	0 row(s) affected	0.001 sec

第二部分 触发器的实现 Trigger in SQL WORKBENCH 5.2.47

一、Trigger 1 商品下单时状态更新

目标： 在订单表上创建新的触发器，当订单定的某产品产品数量大于产品库存，禁止下订单，也就是禁止在订单表中插入记录。

如果商品数量超过库存，则跳过该商品继续下单。否则，更新商品库存，并删除购物车详情中的商品信息。

1. 分析思路

这个状态更新的过程涉及到的表格和操作比较多，具体分析，我梳理出商品下单时状态更新由这几个步骤组成：

1）构造两个新的变量var和mesg来表示购买数量和库存数量

2）使用if…else从句来模拟两种购物可能

3）库存量小于需求量－禁止生成新的订单

4）库存量大于需求量－更新库存并删除购物车中的信息

5）提交业务

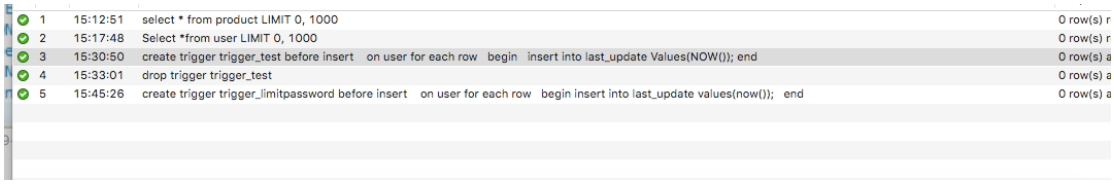
2.代码结构

```
Delimiter $$
create trigger trigger_order
BEFORE INSERT ON payment_description FOR EACH ROW
BEGIN
DECLARE var int;
DECLARE mesg varchar(20);
SELECT product_amount INTO var
FROM product
where product_id=NEW.product_id;

IF var<NEW.purchase_amount
   THEN SELECT XXXX INTO mesg; #禁止在订单表插入记录

ELSE
   UPDATE product SET product_amount=product_amount-NEW.purchase_amount
   where product_id=NEW.product_id;  #更新库存
   DELETE FROM Shopping_Cart_Description WHERE product_id=NEW.product_id;  #删除购物车

END IF;
END $$
```



1	15:12:51	select * from product LIMIT 0, 1000	0 row(s) r
2	15:17:48	Select *from user LIMIT 0, 1000	0 row(s) r
3	15:30:50	create trigger trigger_test before insert on user for each row begin insert into last_update Values(NOW()); end	0 row(s) a
4	15:33:01	drop trigger trigger_test	0 row(s) a
5	15:45:26	create trigger trigger_limitpassword before insert on user for each row begin insert into last_update values(now()); end	0 row(s) a

trigger添加成功

3.调用过程

首先创建 3 个产品信息

product_id	product_name	unit_price	last_update	product_amount
1	computer	2999	2019-06-05 2...	50
2	banana	10	2019-07-05 2...	200
3	book	60	2019-08-05 2...	60
	NULL	NULL	NULL	NULL

再插入一条可以被执行的购物信息和一条不能被执行的购物信息：另一条 VALUES 改成 (3，100)

```
INSERT INTO  payment_description (product_id, purchase_amount)VALUES(1, 49);
```

4. 实验结果

product_id	product_name	unit_price	last_update	product_amount
1	computer	2999	2019-06-05 2...	1
2	banana	10	2019-07-05 2...	200
3	book	60	2019-08-05 2...	60
	NULL	NULL	NULL	NULL

发现 id = 1 的计算机数量减少，但是书的数量没有发生变化，说明 trigger1 可以正常被正常触发。

二、Trigger 2 用户和商品的完整性约束检验

目标：通过触发器检查用户定义完整性约束，本例实现用户录入密码的约束

1. 分析过程

这里由于用户和商品的检验非常相似，我以相比最为复杂的用户密码来进行约束和检验，其他完整性约束都非常类似该案例.

第一步：首先需要确认触发器的类型：应该是插入前就被触发如果密码错误不能被录入系统，所以在选择 before insert 类触发器.

第二步：再分析需要检验的两个指标分别为：

1） 是否同时包含大小写字母和数字

2） 字符串长度是否符合要求

查看了官方文档决定使用 PATINDEX 函数，用来返回字符或者字符串在另一个字符串或者表达式中的起始位置。本 trigger 语法比较简单，使用 if...else 来进行分情况讨论。
第三步：报错的方式选择使用 SQLSTATE '45000'和 message 实现报错

2. 代码结构

```
Delimiter $$
create trigger trigger_limitpassword
before insert on user for each row
begin
  if len(NEW.password)>30 and PATINDEX('%[0-9]%', NEW.password) =0 and PATINDEX('%[a-zA-Z]%', NEW.password) =0
  then signal sqlstate '45000' set MYSQL_ERRNO=30001,
   message_text = 'illegal';
  end if;
End $$
```

3.检验约束

密码设置为一个非法值，既不满足长度要求也不满足同时包含大小写字母和数字的要求

```
insert into user (password)values ('000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
000000000000000')
```

4.实验结果

```
ERROR 30001 (45000): illegal
```

结果表明，当插入不符合规范的 password，系统会自动报错，并 print 出之前设定好的 sqlstate 和 ERROR 代码，trigger 按照预期实现功能。

第三部分 嵌入式 MySQL TRIGGER in PYTHON 3

因为我非计算机专业，相对 java 来说对 python 更为熟悉，所以这次选用 python 的 python 中的 mysql.connector 来进行触发器的操作。

1. 分析思路

首先在 workbench 中使用 drop（）函数撤销掉原有的两个 trigger
对于这两个 trigger，中间的 sql 代码和上一个部分 trigger 是一样的，这里重点叙述和分析 python 的 connector 的链接，我先进行连接:

```
mydb= mysql.connector.connect(host='localhost', user='admin',passwd='admin')
```

初始化完成后，后面使用 mycursor.execute 游标进行数据库指令操作；
这部分最为特殊和困难的部分在于报错的处理

```
try:
  mycursor.execute(trigger2)
except mysql.connector.Error as err:     # Error if raised
  print("Something went wrong: {}".format(err))
```

题目要求使用 System.out.println 提示错误信息，在 python 中相对应的是 print，我使用了 mysql.connector.error 来实现提示错误并用 print 来报错

2. 代码结构

2.1 trigger 1：商品下单时状态更新

```
import mysql.connector
mydb.commit()
mycursor.execute(" drop trigger if exists trigger_order")
trigger1 = ( "Delimiter $$ \n"
"create trigger trigger_order\n"
"BEFORE INSERT ON payment_description FOR EACH ROW\n"
"BEGIN \n"
"DECLARE var int;\n"
"DECLARE mesg varchar(20);\n"
"SELECT product_amount INTO var \n"
"FROM product \n"
"where product_id=NEW.product_id;\n"
"IF var<NEW.purchase_amount \n"
   "THEN SELECT XXXX INTO mesg; #禁止在订单表插入记录\n"
"ELSE\n"
   "UPDATE product SET product_amount=product_amount-NEW.purchase_amount\n"
   "where product_id=NEW.product_id;  #更新库存\n"
   "DELETE FROM Shopping_Cart_Description WHERE product_id=NEW.product_id;  #删除购物车\n"
"END IF; \n"
"END $$\n")
mycursor.execute(trigger1)
mydb.commit()
mydb.close()
```

2.2. trigger 2：用户和商品的完整性约束检查

```
import mysql.connector
mydb.commit()
mycursor.execute((" drop trigger if exists trigger_order")
trigger2 =("Delimiter $$  \n"
"create trigger trigger_limitpassword \n"
"before insert on user for each row\n"
"begin\n"
  "if len(NEW.password)>30 and PATINDEX('%[0-9]%', NEW.password) =0 and PATINDEX('%[a-zA-Z]%', NEW.password)
  "then signal sqlstate '45000'\n"
  "set message_text = 'illegal';\n"
  "end if;\n"
"End $$\n")
try:
  mycursor.execute(trigger2)
except mysql.connector.Error as err:     # Error if raised
  print("Something went wrong: {}".format(err))
mydb.commit()
mydb.close()
```

3. 检验约束

在 anaconda 中使用和上一部分一样的两个例子例子进行测试

```
mycursor.execute("INSERT INTO payment_description(product_id,purchase_amount)VALUES(1,49)")

mycursor.execute("INSERT INTO payment_description(product_id,purchase_amount)VALUES(3,100)")

mycursor.execute("INSERT INTO user(password) VALUES ('000000000000000000000000000000000000000000000000')")
```

前两条指令检验第一个 trigger，第三个指令检验第二个 trigger

4. 实验结果

product_id	product_name	unit_price	last_update	product_amount
1	computer	2999	2019-06-05 2...	1
2	banana	10	2019-07-05 2...	200
3	book	60	2019-08-05 2...	60

发现 id = 1 的计算机数量减少，但是书的数量没有发生变化，说明 trigger1 可以正常被正常触发。

```
ERROR 30001 (45000): illegal
```

同样的系统会自动报错，并 print 出之前设定好的 sqlstate 和 ERROR 代码，trigger 按照预期实现功能。证明嵌入式 SQL 可以正常实现。

第四部分：Procedure 的实现

一、批量购买 Procedure

目标：用户通过存储过程传入所有价格不超过 100 元的商品 X 件，判断是否能完成交易并使用 SQLSTATE '45000'报错

1. 分析过程

题目要求批量购买，即低于某个用户输入的价格的商品都要买 x 件 ，只要库存量够就更新。在用户传入 X 指令后，先分析有哪些商品满足条件，只有都购买才执行操作。

我从而设计了一个 procedure,批量购买的件数为变量，由用户传入 X，使用 if…if…else 从句来保证完整性。针对可能的三种情况：

1） 不存在该商品或库存已经为零，不存在交易的可能 －SQLSTATE '45000'报错

2） 商品库存小于 X，不能完成交易 －SQLSTATE '45000'报错

3） 商品库存不小于 X － 在 product 表中修改库存

2. 代码结构

```
delimiter $$
create procedure buyout(In X int)
Begin
    Declare total_amount INT;
    Select product_amount into total_amount
    From product
    Where unit_price<100;
      IF total_amount IS NULL
        THEN SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An error occurred';
      IF (total_amount-X)<0
        THEN SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An error occurred';
      ELSE
          UPDATE product
          SET total_amount = total_amount  - X
          where unit_price<100;
END
```

3. 调用过程

使用 call 函数调用刚刚设计的 procedure

```
call procedure buyout(100)
```

再次调用 product 表

```
SELECT * from product
```

4．实验结果

Copy Response：

```
ERROR 1644 (45000): illegal
```

product_id	product_name	unit_price	last_update	product_amount
1	computer	2999	2019-06-05 2...	50
2	banana	10	2019-07-05 2...	100
3	book	60	2019-08-05 2...	60
	NULL	NULL	NULL	NULL

操作没有对原始 product 表产生变化，符合实验预期，证明 procedure1 的功能可以正常实现。

二、销售统计 Procedure

1. 分析过程

意识到订单详情表中并没有出现订单总金额，考虑过程为：

1） 构造新变量 tprice = unit_price*product_amount

2） 用 avg（）求平均订单金额

3） 用 where 从句限定 payment 的时间为用户定义的 Y 时刻到 NOW（）调用现在时间

2. 代码结构

```
delimiter $$
create procedure avgpmt(in Y DATETIME)
Begin
  select
  avg(payment_description.tprice=payment_description.purchase_amount*product.unit_price)
  from payment_description,product
  where payment_description.product.id=product.product.id and
  payment_time>='Y' and
  payment_time< now();
END
```

3. 调用过程

插入 3 个订单信息，分别购买 computer, banana, book 的数量为 1，1，1

payment_description 和 product 表格中插入三个用户的数据和三个产品的数据，使用计算器算得在 2019-07-05 日期前平均订单金额为 1504.5

```
call procedure avgpmt (2019-07-05 21:00:00)
```

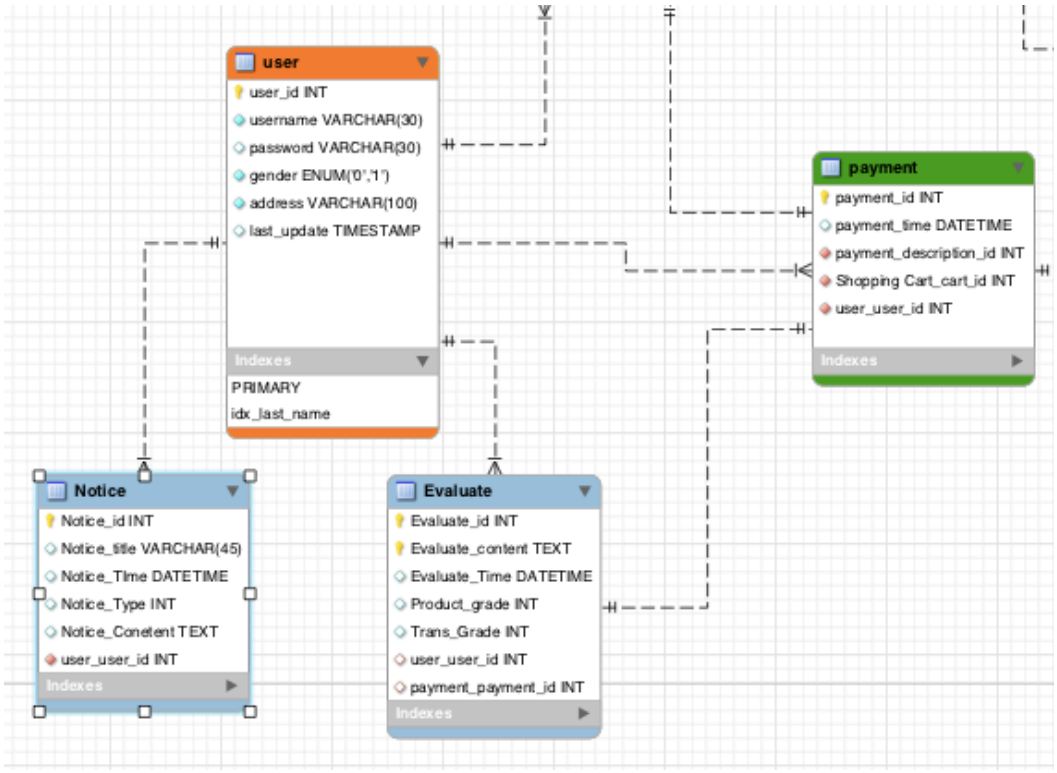
4.实验结果

得到结果 1504.5 **，说明 procedure 可以正常调用。**

第五部分：附加题

由于时间有限但因为现在的功能还比较简单，所以考虑从用户使用体验上进一步完善该系统。因为上面的操作主要在后台进行，考虑在前台插入两个表和用户建立更多的联系，完善用户的购物体验。我又创建了两个新表**交易评价表（Evaluate）和新闻公告表（Notice）**，交易评价表给客户评价订单的权利，可以对产品、物流进行评分并写下自己的评论。新闻公告表是系统向用户发送通知的方式，公告类型主要有两种，1 为广告 2 为产品通知。

新建的两个表格如 E-R 模型所示：



新闻公告表字段属性：

新闻公告表（Notice）

字段名称	类型	约束	描述
Notice_id	INT	NOT NULL	公告编号
Notice_title	VARCHAR(45)	NOT NULL	公告标题
Notice_Type	ENUM(‘1’,’2’)	NOT NULL	公告类型 1 为广告 2 为产品通知
Notice_content	TEXT	NOT NULL	公告内容
User_user_id	INT	NOT NULL	用户编号（外键）
Notice_time	Datetime	NOT NULL	公告时间

交易评价表表字段属性：

交易评价表（Evaluate）

字段名称	类型	约束	描述
Evaluate_id	INT	NOT NULL	评价编号
Evaluate_content	TEXT	NOT NULL	评价内容
Evaluate_Time	DATETIME	NOT NULL	评价时间
Product_Grade	INT	NOT NULL	商品评价
Trans_Grade	INT		发货速度评价
User_user_id	INT	NOT NULL	评价人(外键)
Payment_payment_id	INT	NOT NULL	订单编号(外键)

第六部分：上机作业总结与反思

这次实现的线上购物系统基本实现了一个基本的在线购物系统的需求，用户登录后，可以进行添加购物车，下单等操作，通过分析 3NF 来优化表结果提升 query 效率，通过 trigger 和 procedure 实现表与表的连接和事件的触发，同时还实践了嵌入式 SQL 在 Python 中的应用和具体操作，切实体会到了 SQL 语句的高效和应用之广。

作为文科院系的学生，我感觉这次内容整体非常充实，是对课上知识的极大巩固和补充，中间虽然遇到过一些困难，但是都通过阅读课上 ppt，查阅笔记，阅读 MySQL manual 和看 paper 解决了，这里特别感谢助教的帮助，你们的高效及时的回复让我受益良多，辛苦老师和助教！ 诚然，系统现在还不够完善，同时系统的稳定性也还有提升空间。后续可以考虑再添加一些 trigger 来完善用户与产品之间的联系，也对该系统进行并发控制来解决同时出现大量订单的情况。同时因为现有的功能也比较少，后续也会进一步添加分级会员的机制，并对于附加题中新建立的两个表格，期末前我也会再添加两个 trigger 来管理评论和消息区，优化用户的使用体验。

感谢您的阅读！辛苦～