# MySQL 上机练习

## 说明

本次练习时限为 3 周，目的是让大家体验 MySQL 编程。

有问题随时找助教，助教会尽力为你提供一切帮助。
（包括帮你检查代码中的逻辑错误等等；但不包括直接提供答案，of course~ 😎）

## 作业环境

1. 确保使用的版本是 MySQL 8.0+ 。
2. 下载 `test.zip` 并解压，其中：
   - `README.md` 是运行说明。
   - `answer` 包含答案模板，对于第 x 题，你需要在相应的文件 `scene-x.sql` 中填写你的答案。
   - `public-test` 是一组公开的测试，包含几个测试脚本。
3. 每一题以 `root` 身份访问数据库运行答案与测试（虽然我们不推荐在生产环境中这么做）。
4. 每一题并未指定数据库，但我们创建一个独立的数据库如 `test_student_db` 进行测试。你可以创建其它独立的数据库来导入数据和调试你的 SQL 代码，还可以创建多个数据库来隔离题目。

请根据 `README.md` 中的说明，先运行加载数据脚本，然后运行你的答案，最后运行测试。

> ⚠ **Warning**
>
> 注意，`README` 中的 `source` 命令使用的是相对路径，所以需要确保终端运行在当前目录，或者将其改为绝对路径。

## 提交

将你的 `answer` 文件夹压缩到一个 ZIP 文件，命名为 `学号-姓名.zip`，其内容为包含所有回答的 `answer` 文件夹。

> ⚠ **Warning**
>
> 每一题的代码文件，文件名一定要与题目对应。

# 背景

某品牌的一家国内代理商需要管理他们的订单。在管理系统立项时，决定将主要逻辑写在数据库端；作为该代理商的后端工程师，你需要编写一些 SQL 脚本来满足需求。

其中用到的表有：

```
CREATE TABLE Product(
    id              VARCHAR(15) NOT NULL,
    pname           VARCHAR(255) NOT NULL,
    price           INT UNSIGNED NOT NULL,
    stock           INT UNSIGNED NOT NULL,
    manufacturer    VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE Customer(
    id              VARCHAR(15) NOT NULL,
    cname           VARCHAR(255) NOT NULL,
    city            VARCHAR(255) NOT NULL,
    phone_number    VARCHAR(31) NOT NULL,
    balance         INT UNSIGNED NOT NULL,
    credit          INT UNSIGNED NOT NULL DEFAULT 1000,
    PRIMARY KEY (id)
);

CREATE TABLE ProductOrder(
    id              VARCHAR(15) NOT NULL,
    create_date     DATE NOT NULL,
    customer_id     VARCHAR(15) NOT NULL,
    product_id      VARCHAR(15) NOT NULL,
    quantity        INT UNSIGNED NOT NULL,
    is_filled       BOOLEAN NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (customer_id) REFERENCES Customer(id),
    FOREIGN KEY (product_id) REFERENCES Product(id)
);
```

# 场景1 存储过程与游标（20分）

这是一道综合性的题目，需要你灵活运用 SQL 语言中的查询、游标、条件判断、循环等语句。

经理准备开展优惠活动，他想预计如果采取折扣、过去的一段时间内每个客户的总花销是多少。请为他编写一个存储过程 `EvaluateDiscountedSales`，请仔细阅读以下需求。

## 存储过程定义

需要定义的存储过程名和参数为：

```
EvaluateDiscountedSales(
    p_date_1 DATE,
    p_date_2 DATE,
    p_price_threshold INT,
    p_cost_threshold INT,
    p_city_1 VARCHAR(255),
    p_city_2 VARCHAR(255),
    p_manufacturer_1 VARCHAR(255),
    p_manufacturer_2 VARCHAR(255)
)
```

## 性能需求

在测试时，我们使用的产品数 $\leq 100$，客户数 $\leq 100$，订单数 $\leq 10^5$。

为了不长时间占用机器，我们在测试时预留 10 分钟的运行时限（经试验，性能较好的写法运行时间不会超过 2 分钟）。如超时会扣除 20% 的分数。

## 功能需求

简而言之，经理想要知道日期 `>= p_date_2` 的每个客户的总花销。根据客户在日期 `<= p_date_1` 的订单情况，这一花销可能会有折扣。

具体而言，折扣共两种，它们相乘计算。

- **大客户折扣：**
  - **资格：**
    - 经理认为客户若购买高价商品较多，则应视为大客户，给予折扣。
    - 对于某一客户，在日期 `<= p_date_1` 订购的产品中，单价 `> p_price_threshold` 的所有产品的订单总额 T `> p_cost_threshold`，则该客户被列为 *大客户*，该客户在日期 `>= p_date_2` 订购的 *所有产品* 获得折扣。
    - 城市位于 `p_city_1` 或 `p_city_2` 的客户不参与该折扣。
  - **折扣：** 分为两个等级，对于所有大客户：

        - 先按客户信用（`credit`）降序、后按上述总金额 T 降序排序；
        - 前10名（使用 `RANK` 计算，包括并列第10名）取9折，其余取95折。
- **多次购买折扣：**
    - **资格：**
        - 经理认为客户若在某一产品上订购较多，则在同种产品上应该获得折扣。
        - 对于某一客户，在日期 `<= p_date_1` 订购过产品，则该客户在日期 `>= p_date_2` 订购的 *同种产品* 获得折扣 。
        - 生产商 `p_manufacturer_1` 和 `p_manufacturer_2` 的产品不参与该折扣。
    - **折扣：** 分为三个等级：
        - 在日期 `<= p_date_1` 订购过1次及以上 *且* 总金额大于等于1000的，该产品在日期 `>= p_date_2` 打98折；
        - 在日期 `<= p_date_1` 订购过2次及以上 *且* 总金额大于等于3000的，该产品在日期 `>= p_date_2` 打95折；
        - 在日期 `<= p_date_1` 订购过3次及以上 *且* 总金额大于等于6000的，该产品在日期 `>= p_date_2` 打9折。

为了检查结果，你的存储过程需要创建2个新表 `BigCustomer` 和 `NormalCustomer` 用于存放结果：

- 列名均为客户名 `cname VARCHAR(255)` 和总花销（向下取整） `total_cost INT`；
- 将 *大客户* 及其日期 `>= p_date_2` 的总花销放到 `BigCustomer` 表；
- 其他客户及其日期 `>= p_date_2` 的总花销放到 `NormalCustomer` 表；
- 两表总共应包含所有客户。

## 实现要求

- 不考虑订单是否已交付（`is_filled`），全部计入花销；
- 因为出题时的简化，使用了 `INT` 作为存储客户余额的单位 。为了避免精度问题，计算花销请全程按 `DOUBLE` 或 `DECIMAL` 计算，最后 **向下取整** 。
- 必须用到游标，即使查询的速度比游标更快（出于性能考虑，我们不推荐在生产环境使用游标。但出于练习原因，需在本题至少一个步骤步中使用游标，未使用游标会扣除本题 20% 的分数）；
- 可以在存储过程中创建临时表，但在存储过程结束后，除了新增的 `BigCustomer` 和 `NormalCustomer` 表，数据库不应多出其他表。

## 提交

将你的存储过程填写到 `scene-1.sql` 中。

# 场景2 添加索引（20分）

作为数据库工程师，你想要优化存储过程的执行效率。尝试在某些表的合适的字段上添加索引（不超过2个），使上一题存储过程的执行时间至少缩短 50%.

## 要求

上题的存储过程应至少通过一个测试用例，且相比于未添加索引的情况，在添加索引（不超过2个）后，该存储过程在该测试用例上的执行时间缩短了至少 50%.

## 提交

将设置索引的语句填写到 `scene-2.sql` 中。文件应只包含若干条索引创建语句，即 `CREATE INDEX ... ;`，执行后添加的索引总数不超过 2 条。

# 场景3 触发器---取消订单（20分）

客户有可能因为退货等原因取消订单。请你编写一个触发器，在有订单被删除时自动执行合适的操作。

## 要求

触发器名为 `RefundAtCancel`，在 `ProductOrder` 表中有订单被删除时，触发操作：

- 如果订单未交付（ `is_filled = false` ）则无需操作；
- 如果订单已交付（ `is_filled = true` ），则将订单数量退回其产品的库存中，并将订单总金额退回该客户的余额中。

## 提交

将触发器填写到 `scene-3.sql` 中。

# 场景4.1 触发器---审计（10分）

为了定期审计订单表的改动是否存在异常，经理决定引入一个审计日志表，以在订单表发生更新时自动记录审计日志：

```
CREATE TABLE OrderAudit(
    change_time         TIMESTAMP NOT NULL,
    operator            VARCHAR(255) NOT NULL,
    old_order_id        VARCHAR(15),
    old_create_date     DATE,
    old_customer_id     VARCHAR(15),
    old_product_id      VARCHAR(15),
    old_quantity        INT,
    old_is_filled       BOOLEAN,
    new_order_id        VARCHAR(15),
    new_create_date     DATE,
    new_customer_id     VARCHAR(15),
    new_product_id      VARCHAR(15),
    new_quantity        INT,
    new_is_filled       BOOLEAN
);
```

请你编写一个触发器来满足自动记录的需求。

## 要求

触发器名为 `LogOrderChange`，在订单表发生更新操作时自动往 `OrderAudit` 表中写入审计日志，每一列的含义为：

- `change_time` 为当前时间戳。只会检查日期；
- `operator` 为触发变更的客户全称，即 `客户名@IP地址` ；
- `old_xxx` 为旧的订单属性，`new_xxx` 为新的订单属性，与 `ProductOrder` 表的属性应当一一对应。

## 提示

MySQL 触发器中的 `OLD` 与 `NEW` 变量可以反映发生变化的行。请自行查阅文档。

## 提交

将触发器填写到 `scene-4.1.sql` 中。

# 场景4.2 权限管理（10分）

现有一个销售经理（或一个销售管理程序）需要对审计日志进行管理。请你模拟这一场景编写一些与权限相关的 SQL 语句。

## 要求

分以下 3 个小题，编写 SQL 语句：

1. 请创建一个客户 `SalesManager` ，登录地址为 `localhost` ，密码为 `123456` ，并授予 `OrderAudit` 表的所有权限，且这些权限可以授予其他客户。
2. 请以 `SalesManager` 的视角将查询权限授予 `Alice` ， 登录地址为 `localhost` （假设执行前 `Alice` 客户已存在）。
3. 请你以 `Alice` 的视角写一个查询，查找审计日志中只有数量被更新的订单记录。查询结果包含审计日志表 `OrderAudit` 的所有列，列名相同。

## 提交

以上 3 个小题的答案请分别填写到 `scene-4.2.1.sql` , `scene-4.2.2.sql` , `scene-4.2.3.sql` 中。第一个小题应只有 2 条 SQL 语句，而后两步应各只有 1 条 SQL 语句（语句以分号 `;` 结尾）。

# 场景5 事务、保存点与回滚（20分）

为了更好地管理优惠，经理决定引入一个优惠券表

```
CREATE TABLE Coupon(
    customer_id     VARCHAR(15) NOT NULL,
    discount        DOUBLE NOT NULL,
);
```

其中 `discount` 是折扣后的百分比，例如 95 折是 `0.95`。

引入优惠券表后，在交付订单（即扣除余额和库存）时，订单的产品数量、产品的价格与库存、客户的余额存在一致性关系。在生产环境中，每张表的查询和更新操作都有可能由于未知原因失败。为了避免不可预料的失败对交付操作造成影响，需要用到事务。

## 要求

创建一个包含事务的存储过程，存储过程名和参数为：

```
CreateAndFillOrder(
    p_order_id      VARCHAR(15),
    p_customer_id   VARCHAR(15),
    p_product_id    VARCHAR(15),
    p_quantity      INT UNSIGNED
)
```

该过程执行一段创建订单并尝试交付的事务

1. 根据参数在 `ProductOrder` 表中创建一个订单，设为未交付状态；
2. 尝试扣除库存；
3. 尝试扣除余额，有以下情况：
    - 若有优惠券，尝试同时扣取折扣力度最大的优惠券和优惠后的总价；
    - 如果上述情况失败，尝试扣取全价；
    - 如果上述情况失败，尝试从信用(credit)抵扣不足的部分，且不能使用优惠券；
4. 2、3步同时成功，则将订单设为已交付；否则，回退到未交付的状态。

**在测试中，我们会设置一些“意外”（用来模拟网络出了问题或系统故障），使得库存/余额/优惠券充足也有可能更新失败。因此，不仅需要数值判断，还需要使用保存点和回滚来确保这些步骤的完整性。**

我们约定“意外”引发的错误一定是 `SQLSTATE 45000`，请查阅相关资料。

> **Info**
>
> 某些列已设为无符号类型 `UNSIGNED` ，因此尝试将这些设为小于 0 将直接引发异常。如果通过 `DECLARE CONTINUE HANDLER` 来处理异常，你可以直接判断这一异常而无需判断数值。当然，判断数值也是可以的。

## 提示

怎样发现每一条SQL语句是否出现异常？请查阅相关资料。

## 提交

将带有事务的存储过程填写到在 `scene-5.sql` 中。