京东连锁商店数据库

1 设计思路综述

1.1 需求的分析

在商店层次，难点在于不同的商店会有不同的商品，也可能会有同一商品。且还要做到每个商店都有自己的商品库存，每个商店都需要决定何时再订购、订购的价格以及订购数量。

在商品层次，难点在于不同商店的同一商品可能会有不同的进货价格和销售价格。且不同的时间，这个也可能发生变化。

在客户层次，难点在于顾客可以通过加入会员来存储相关的销售信息。

这里为了实现上述关键点和难点，创新地设计了和商店商品客户三者有关的销售记录实体，还有和商店商品两者有关的进货记录实体。这里每个商店的库存只和商店商品有关，且属性较少，所以这里我们把它作为商店和商品的一个关系来处理。这样设计的数据库既能直观简洁地解决问题，又不冗余，且我加入了很多相关约束和完整性，保证了数据库的安全和完整。

总之，该数据库能够高效安全地跟踪每家商店的库存、顾客购买情况、商店的销售历史等。该数据库系统能够高效地提供访问数据库来记录销售、启动再订购、处理到达的新订单等。

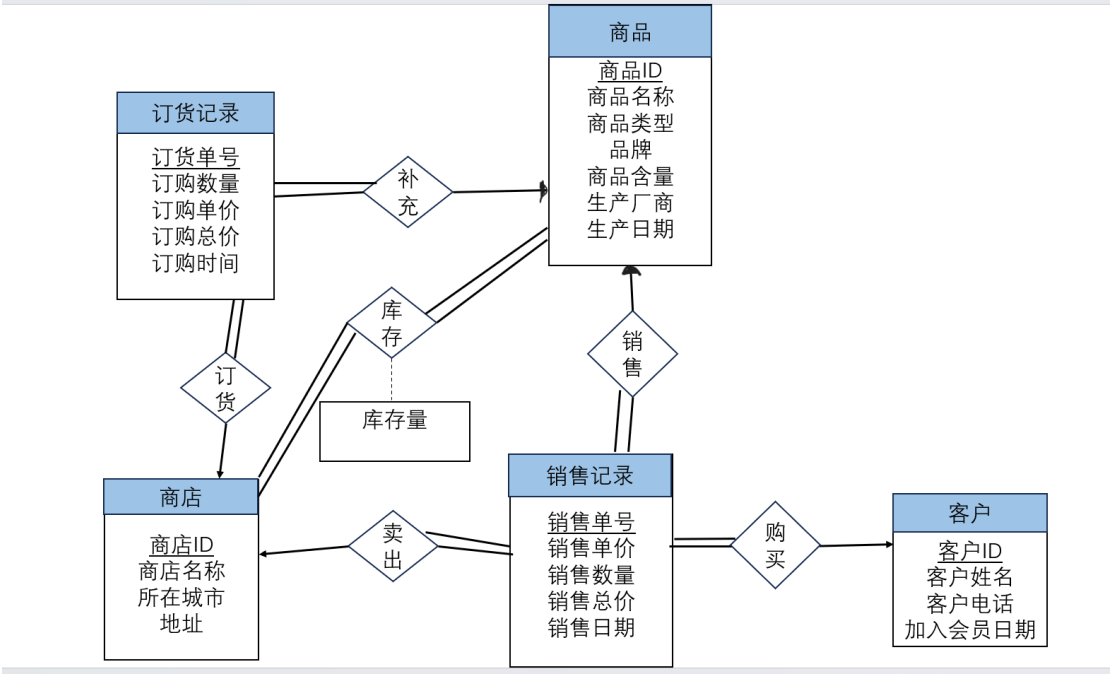
1.2 一些优化的亮点

为了系统的完整性和安全性，这里加入了多个 check 约束去保证，还有一些安全性检测，级联删除的规范啊等等。为了方便做一些查询，我加入了一些索引。

为了使得数据有趣，我的数据是以真实的京东旗舰下的商店而拓开的，并且保证了足够多样性的数据去操作。

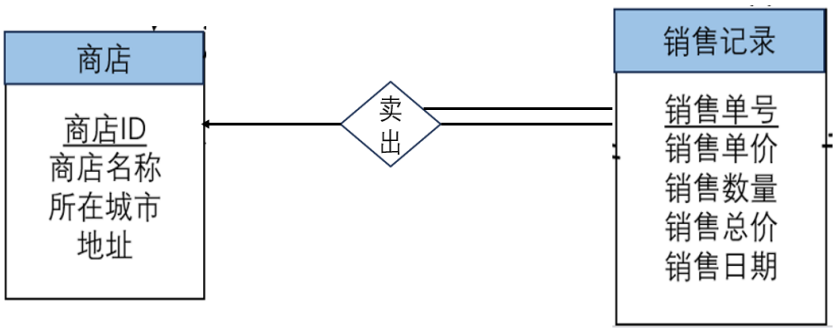
2 E-R 图

2.1 全局 E-R 图

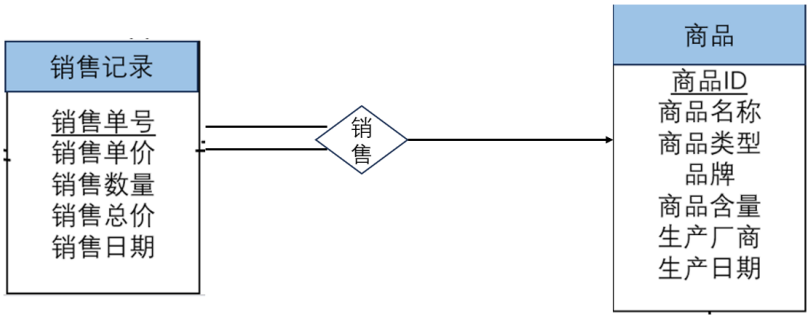


这里，我创立了总共五个实体。三个实体，商店、商品、客户是确定的，毋庸置疑的。由于该数据库需要跟踪每家商店的库存、顾客购买情况、商店的销售历史等。而且每家店的同一种商品，销售价格和进货价格可以不一样。介于以上情况，如果我们把销售情况和进货情况设为关系，虽然也可以实现要求，但是不方便，不够直观和好用。所以我的思路是把销售情况和进货情况作为实体，这样，所有查询记录，做触发器，程序，函数等操作，都会变得简单和直观。每个实体的对应关系（一对多还是多对多，全部不全部参与），是基于我设计的数据库和我设计的数据来确定的。我将库存作为了商品和商店的关系，将库存量作为关系的属性。

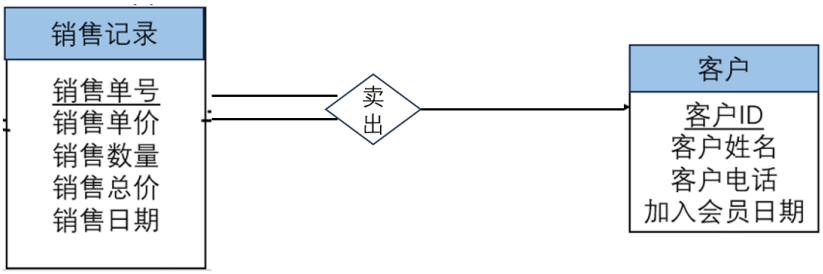
2.2 局部 E-R 图



商店和销售记录，每个商店可以有多个销售记录，每个销售记录只能属于一个商店，所以商店和销售记录是一对多的关系。且每个商店不一定有销售记录，比如才新开的商店，而销售记录是一定有一个商店与它对应。所以销售记录是全部参与的，商店不一定全部参与。它们的关系应该是卖出的关系。

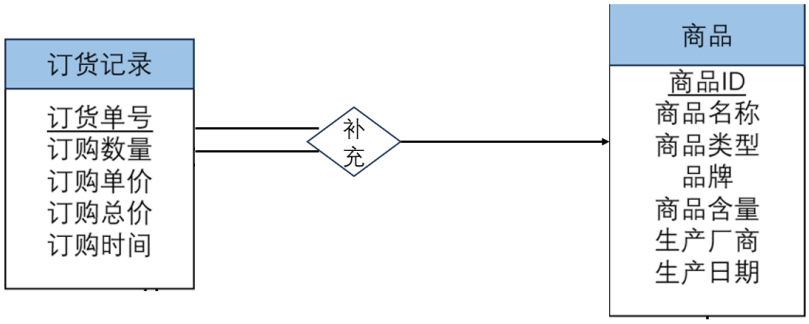


销售记录和商品，这里我们每个销售记录单上只写一种商品，每个商品也可以多次被卖。所以销售记录和商品是多对一的关系。且每个销售记录一定有对应的商品，而每个商品不是一定有销售记录，所以销售记录是全部参与，商品不一定全部参与。它们的关系是销售的关系。

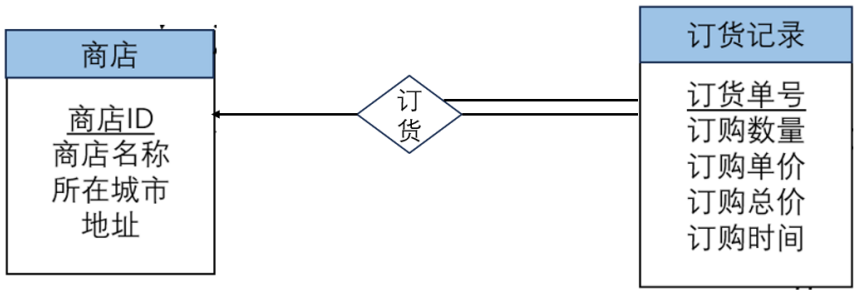


销售记录和客户，一个销售记录只能对应一个客户，而每个客户可以有多个销

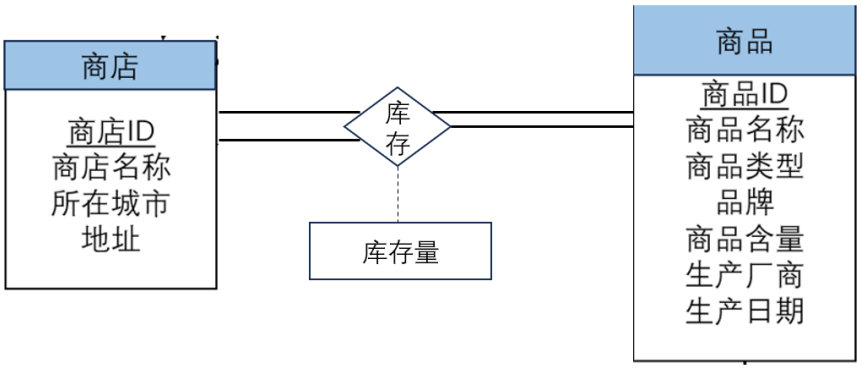
售记录。且销售记录一定有客户对应，而客户不一定有销售记录。所以销售记录是全部参与，客户不一定。它们之间是卖出的关系。



订货记录和商品，这里每个订货记录单只写一种商品，每个商品可以有多个订货记录，所以是多对一的关系。订货记录一定有商品对应，而商品不一定会有订货记录。所以订货记录全部参与，商品不一定。它们之间是补充关系。



商店和订货记录，每个商店有多个订货记录，每个订货记录只属于一个商店，所以是一对多的关系。且每个订货记录一定有商店与它对应，商店不一定，比如新开的商店。所以订货记录是全部参与，商店不一定是全部参与。它们之间是订货关系。



商店和商品，每个商店有多种商品，每个商品也可以有多个商店库存它卖它，所以是多对多的关系。这里认为，即使是新开的商店，也不能完全没有商品，所以商店至少一定会有商品。这里认为，商品一定会有商店库存它卖它，所以认为它们都是全部参与。它们之间的关系是库存关系，库存关系有一个库存量的属性，去记录每个商店的每个商品的库存量。

3 关系模式

把一对多，一对一的关系的属性都按照转化关系模式的规则，写入一的那方的实体。最终按照转化关系模式的规则，我们得到关系模式。

客户表（客户 ID，客户姓名，客户电话，加入会员日期）
商品表（商品 ID，商品名称，商品类型，品牌，商品含量，生产厂商，生产日期）
商店表（商店 ID，商店名称，所在城市，地址）
订货记录表（订货单号，商店 ID，商品 ID，订购数量，订购单价，订购总价，订购时间）
销售记录表（销售单号，商店 ID，商品 ID，客户 ID，销售单价，销售数量，销售总价，销售日期）
库存表（商品 ID，商店 ID，库存量）

4 物理结构设计

表 4-1 客户表 Client

列名	数据类型	是否可为空	键	说明
client_id	Char(4)	Not null	主键	顾客ID
client_name	Char(20)	Not null		顾客姓名
mobile	Char(11)	1	唯一键	顾客手机号

一般商店都会存储会员的手机号，并且每个人的手机号是唯一的。

表 4-2 商店表 Store

列名	数据类型	是否可为空	键	说明
store_id	Char(7)	Not null	主键	商店 ID
store_name	Char(30)	Not null		商店名称
city	Char(16)	Not null		所在城市
address	Char(30)	Not null		商店的具体地址

因为是实体商店，所以门店的具体地址也很重要，所以这里要有这个属性。

表 4-2 商品表 Goods

列名	数据类型	是否可为空	键	说明
goods_id	Char(10)	Not null	主键	商品 ID
goods_name	Char(30)	Not null		商品名称
goods_type	Char(20)	Not null		商品类型
goods_brand	Char(20)	null		商品品牌
goods_units	Char(26)	Not null		一份商品的单位
goods_yield	Char(40)	Not null		生产厂商
goods_date	date	Not null		生产日期

商品的类型，品牌一般都要说明。这里 goods_units 是商品一份的单位，比如大米一斤，食用油一桶等量化单位。商品都要有生产厂商和生产日期表明，所以加入了这些属性。

表 4-2 销售记录表 SalesOrders

列名	数据类型	是否可为空	键	说明
Sale_id	Char(10)	Not null	主键	销售单 ID
Goods_id	Char(10)	Not null	外码	商品 ID
Client_id	Char(4)	Not null	外码	客户 ID
sale_price	decimal(8,2)	Not null		销售单价
store_id	Char(7)	Not null	外码	商店 ID
sale_num	int	Not null		销售总数
sale_sumprice	Decimal(10,2)	Not null		销售总金额
sale_date	datetime	Not null		销售日期

这里有三个外码，用于去查询对应的商店，商品，客户的销售记录。每次买一个商品，可能不止买一个，所以这里加入了销售数量，销售单价和销售总价。有了销售总价，我们统计商店或者客户的销售总额就会非常方便。当然，销售日期也很重要。

表 4-2 进货记录表 Ordering

列名	数据类型	是否可为空	键	说明
ordering_id	Char(10)	Not null	主键	进货单 ID
goods_id	Char(10)	Not null	外码	商品 ID
Store_id	Char(7)	Not null	外码	商店 ID
ordering_price	Decimal(8,2)	Not null		进货单价
ordering_num	int	Not null		进货数量
ordering_sum	Decimal(10,2)	Not null		进货总价
ordering_date	datetime	Not null		进货日期

同样的，这里有两个外键，用于去查询。同样，一次进货进一种商品，可能不止进货一个，所以有进货单价，数量和总价。进货日期也很重要。

表 4-2 库存表 Stock

列名	数据类型	是否可为空	键	说明
store_id	Char(7)	Not null	外码	商店 ID
goods_id	Char(10)	Not null	外码	商品 ID
stock_num	int	null		库存量

这里某件商品可能没有库存记录，所以库存量可以为 null。

5 数据库实现

5.1 建立数据库

我们首先建立一个合适名字的数据库

```
mysql> create database JingDongStore
    -> ;
```

```
mysql> use JingDongStore;
Database changed
```

5.2 建立表的结构

然后根据前面的关系模式和 E-R 图建立表的结构，这里在 Client 表中，我建立了检查电话号码的 check 约束。Goods 表中，我建立了检查生产日期的 check 约束。在 SalesOrders 表中，我建立了三个 check 约束，并且对外键加入了级联删除的规范，防止数据库发生错误。Stock 表和 Ordering 表同样做出相应的规范处理。这里 SalesOrders 表和 Ordering 表日期的单位用的是 datetime，因为进货和销售理应记录具体的时间。但由于数据太多，在插入数据时，我并没有输入具体的时间。但理应是 datetime。

```
mysql> CREATE TABLE Client (
    ->     client_id CHAR(4) NOT NULL,
    ->     client_name CHAR(20) NOT NULL,
    ->     mobile CHAR(11) NOT NULL,
    ->     PRIMARY KEY (client_id),
    ->     UNIQUE (mobile),
    ->     CHECK (CHAR_LENGTH(mobile) = 11 AND mobile REGEXP '^[0-9]+$')
    -> );
Query OK, 0 rows affected (0.10 sec)
```

```
mysql> CREATE TABLE Goods (
    ->     goods_id CHAR(10) NOT NULL,
    ->     goods_name CHAR(30) NOT NULL,
    ->     goods_type CHAR(20) NOT NULL,
    ->     goods_brand CHAR(20),
    ->     goods_units CHAR(26) NOT NULL,
    ->     goods_yield CHAR(40) NOT NULL,
    ->     goods_date DATE NOT NULL,
    ->     PRIMARY KEY (goods_id),
    ->     CHECK (goods_date >= '2000-01-01')
    -> );
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> CREATE TABLE Store (
    ->     store_id CHAR(7) NOT NULL,
    ->     store_name CHAR(30) NOT NULL,
    ->     city CHAR(16) NOT NULL,
    ->     address CHAR(30) NOT NULL,
    ->     PRIMARY KEY (store_id)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> CREATE TABLE SalesOrders (
    ->     sale_id CHAR(10) NOT NULL,
    ->     goods_id CHAR(10) NOT NULL,
    ->     client_id CHAR(4) NOT NULL,
    ->     sale_price DECIMAL(8,2) NOT NULL,
    ->     store_id CHAR(7) NOT NULL,
    ->     sale_num INT NOT NULL,
    ->     sale_sumprice DECIMAL(10,2) NOT NULL,
    ->     sale_date DATETIME NOT NULL,
    ->     PRIMARY KEY (sale_id),
    ->     FOREIGN KEY (goods_id) REFERENCES Goods(goods_id)
    ->         ON DELETE CASCADE ON UPDATE CASCADE,
    ->     FOREIGN KEY (store_id) REFERENCES Store(store_id)
    ->         ON DELETE CASCADE ON UPDATE CASCADE,
    ->     FOREIGN KEY (client_id) REFERENCES Client(client_id)
    ->         ON DELETE CASCADE ON UPDATE CASCADE,
    ->     CHECK (sale_price > 0),
    ->     CHECK (sale_num > 0),
    ->     CHECK (sale_sumprice = sale_price * sale_num)
    -> );
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> CREATE TABLE Ordering (
    ->     ordering_id CHAR(10) NOT NULL,
    ->     goods_id CHAR(10) NOT NULL,
    ->     store_id CHAR(7) NOT NULL,
    ->     ordering_price DECIMAL(8,2) NOT NULL,
    ->     ordering_num INT NOT NULL,
    ->     ordering_sum DECIMAL(10,2) NOT NULL,
    ->     ordering_date DATETIME NOT NULL,
    ->     PRIMARY KEY (ordering_id),
    ->     FOREIGN KEY (goods_id) REFERENCES Goods(goods_id)
    ->         ON DELETE CASCADE ON UPDATE CASCADE,
    ->     FOREIGN KEY (store_id) REFERENCES Store(store_id)
    ->         ON DELETE CASCADE ON UPDATE CASCADE,
    ->     CHECK (ordering_price > 0),
    ->     CHECK (ordering_num > 0),
    ->     CHECK (ordering_sum = ordering_price * ordering_num)
    -> );
Query OK, 0 rows affected (0.07 sec)
```

```
mysql> CREATE TABLE Stock (
    ->     store_id CHAR(7) NOT NULL,
    ->     goods_id CHAR(10) NOT NULL,
    ->     stock_num INT,
    ->     FOREIGN KEY (store_id) REFERENCES Store(store_id)
    ->         ON DELETE CASCADE ON UPDATE CASCADE,
    ->     FOREIGN KEY (goods_id) REFERENCES Goods(goods_id)
    ->         ON DELETE CASCADE ON UPDATE CASCADE,
    ->     PRIMARY KEY (store_id, goods_id),
    ->     CHECK (stock_num >= 0)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

这里我添加了 SalesOrders 的一个索引和 Ordering 的一个索引，便于去查询和商店有关的销售记录。

```
mysql> -- 添加索引优化
mysql> CREATE INDEX idx_salesorders_store_date ON SalesOrders(store_id, sale_date);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_ordering_store_date ON Ordering(store_id, ordering_date);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

5.3 插入数据

插入数据这里，我们商店插入了三个城市的各五个商店，都是京东旗舰下的，都是真实存在的。地点和名称都是完全真实的。

```
mysql> INSERT INTO Store (store_id, store_name, city, address) VALUES
    -> -- 北京
    -> ('0000001', '京东七鲜超市(王府井店)', '北京', '东城区王府井大街'),
    -> ('0000002', '京东之家(西单店)', '北京', '西城区西单大悦城'),
    -> ('0000003', '京东电器(中关村店)', '北京', '海淀区中关村广场'),
    -> ('0000004', '京东便利店(国贸店)', '北京', '朝阳区国贸中心'),
    -> ('0000005', '京东生鲜(朝阳大悦城店)', '北京', '朝阳区朝阳大悦城'),
    ->
    -> -- 深圳
    -> ('0000006', '京东七鲜超市(华强北店)', '深圳', '福田区华强北路'),
    -> ('0000007', '京东之家(南山店)', '深圳', '南山区海岸城'),
    -> ('0000008', '京东电器(宝安店)', '深圳', '宝安区宝安中心'),
    -> ('0000009', '京东便利店(罗湖店)', '深圳', '罗湖区东门步行街'),
    -> ('0000010', '京东生鲜(龙华店)', '深圳', '龙华区龙华街道'),
    ->
    -> -- 哈尔滨
    -> ('0000011', '京东七鲜超市(中央大街店)', '哈尔滨', '道里区中央大街'),
    -> ('0000012', '京东之家(哈西店)', '哈尔滨', '南岗区哈西万达广场'),
    -> ('0000013', '京东电器(江北店)', '哈尔滨', '松北区世茂大道'),
    -> ('0000014', '京东便利店(学府路店)', '哈尔滨', '南岗区学府路'),
    -> ('0000015', '京东生鲜(道外店)', '哈尔滨', '道外区南直路');
Query OK, 15 rows affected (0.00 sec)
Records: 15  Duplicates: 0  Warnings: 0
```

客户这里我们把 0000 匿名作为非会员，从 0001 开始作为会员，并且有比较真实的姓名和电话号码。共有 35 个客户，是足够我们操作的。

```
mysql> INSERT INTO Client (client_id, client_name, mobile) VALUES
    -> ('0000', '匿名', '00000000000'),
    -> ('0001', '陈泓平', '13912345678'),
    -> ('0002', '李进德', '13823456789'),
    -> ('0003', '王志强', '13734567890'),
    -> ('0004', '赵子龙', '13645678901'),
    -> ('0005', '孙小雅', '13556789012'),
    -> ('0006', '周国华', '13467890123'),
    -> ('0007', '吴家伟', '13378901234'),
    -> ('0008', '郑丽芳', '13289012345'),
    -> ('0009', '冯宏伟', '13190123456'),
    -> ('0010', '陈思远', '13001234567'),
    -> ('0011', '褚海龙', '13922334455'),
    -> ('0012', '卫文彬', '13833445566'),
    -> ('0013', '蒋天成', '13744556677'),
    -> ('0014', '沈燕妮', '13655667788'),
    -> ('0015', '韩建明', '13566778899'),
    -> ('0016', '杨志杰', '13477889900'),
    -> ('0017', '朱伟峰', '13388990011'),
    -> ('0018', '秦浩然', '13299001122'),
    -> ('0019', '尤天宇', '13110112233'),
    -> ('0020', '许德华', '13021223344'),
    -> ('0021', '何家兴', '13932334455'),
    -> ('0022', '吕松林', '13843445566'),
    -> ('0023', '施耀东', '13754556677'),
    -> ('0024', '张智勇', '13665667788'),
    -> ('0025', '孔美玲', '13576778899'),
    -> ('0026', '曹晓东', '13487889900'),
    -> ('0027', '严伟峰', '13398990011'),
    -> ('0028', '华志强', '13209001122'),
    -> ('0029', '金建明', '13120112233'),
    -> ('0030', '魏浩然', '13031223344'),
    -> ('0031', '陶德华', '13942334455'),
    -> ('0032', '姜志杰', '13853445566'),
    -> ('0033', '戚天成', '13764556677'),
    -> ('0034', '谢伟峰', '13675667788');
Query OK, 35 rows affected (0.01 sec)
Records: 35  Duplicates: 0  Warnings: 0
```

对于商品，我选择了各式各样的商品，商品的名称都很真实，出产厂商和品牌都是真实的和合理的。共插入了 50 条商品信息，是完全足够我们操作的。

```
mysql> INSERT INTO Goods (goods_id, goods_name, goods_type, goods_brand, goods_units, goods_yield, goods_date) VALUES
    -> ('G00001', '云南新鲜生菜', '蔬菜', '华润万家', '约400g', '云南华润万家有限公司', '2023-07-15'),
    -> ('G00002', '四川红富士苹果', '水果', '京东生鲜', '约1kg', '四川京东生鲜有限公司', '2023-06-10'),
    -> ('G00003', '黑龙江有机大米', '粮油', '五常稻花香', '5kg', '黑龙江五常稻花香有限公司', '2023-05-20'),
    -> ('G00004', '新疆特级大枣', '坚果', '西域香', '500g', '新疆西域香农产品有限公司', '2023-04-25'),
    -> ('G00005', '山东红心火龙果', '水果', '果鲜生', '约2个', '山东果鲜生有限公司', '2023-08-01'),
    -> ('G00006', '宁夏枸杞', '保健品', '百草味', '250g', '宁夏百草味有限公司', '2023-03-18'),
    -> ('G00007', '进口牛奶', '乳制品', '德运', '1L', '澳大利亚德运有限公司', '2023-07-05'),
    -> ('G00008', '西湖龙井茶', '茶叶', '龙井村', '250g', '浙江龙井村有限公司', '2023-04-30'),
    -> ('G00009', '东北野生松茸', '菌菇', '东北老林', '500g', '黑龙江东北老林有限公司', '2023-06-12'),
    -> ('G00010', '广西沃柑', '水果', '新希望', '约5个', '广西新希望有限公司', '2023-05-22'),
    -> ('G00011', '安溪铁观音', '茶叶', '八马茶业', '250g', '福建八马茶业有限公司', '2023-05-15'),
    -> ('G00012', '青海牦牛肉', '肉类', '三江源', '500g', '青海三江源有限公司', '2023-06-28'),
    -> ('G00013', '内蒙古风干牛肉', '肉类', '蒙牛', '250g', '内蒙古蒙牛有限公司', '2023-03-30'),
    -> ('G00014', '海南椰子', '水果', '椰岛', '3个', '海南椰岛有限公司', '2023-07-18'),
    -> ('G00015', '贵州茅台酒', '酒水', '茅台', '500ml', '贵州茅台酒厂', '2023-01-15'),
    -> ('G00016', '进口橄榄油', '油类', '贝蒂斯', '1L', '西班牙贝蒂斯有限公司', '2023-02-22'),
    -> ('G00017', '国产蜂蜜', '调味品', '百花', '500g', '北京百花蜂业有限公司', '2023-04-01'),
    -> ('G00018', '有机鸡蛋', '蛋类', '农夫山泉', '30个', '浙江农夫山泉有限公司', '2023-05-28'),
    -> ('G00019', '新疆库尔勒香梨', '水果', '果园', '约6个', '新疆果园有限公司', '2023-07-11'),
    -> ('G00020', '进口红酒', '酒水', '拉菲', '750ml', '法国拉菲酒庄', '2023-03-05'),
    -> ('G00021', '苹果 iPhone 14', '手机', 'Apple', '1台', 'Apple Inc.', '2023-01-10'),
    -> ('G00022', '小米 12', '手机', '小米', '1台', '小米科技有限公司', '2023-02-15'),
    -> ('G00023', '华为 P50', '手机', '华为', '1台', '华为技术有限公司', '2023-03-20'),
    -> ('G00024', '三星 Galaxy S22', '手机', '三星', '1台', '三星电子有限公司', '2023-04-25'),
    -> ('G00025', 'OPPO Find X5', '手机', 'OPPO', '1台', 'OPPO电子有限公司', '2023-05-30'),
    -> ('G00026', '索尼 Bravia 电视', '电器', '索尼', '1台', '索尼公司', '2023-01-05'),
    -> ('G00027', '飞利浦 电动剃须刀', '电器', '飞利浦', '1台', '飞利浦公司', '2023-02-20'),
```

```
mysql> select * from Goods;
+----------+--------------------+------------+-------------+-------------+--------------------------+------------+
| goods_id | goods_name         | goods_type | goods_brand | goods_units | goods_yield              | goods_date |
+----------+--------------------+------------+-------------+-------------+--------------------------+------------+
| G00001   | 云南新鲜生菜       | 蔬菜       | 华润万家    | 约400g      | 云南华润万家有限公司     | 2023-07-15 |
| G00002   | 四川红富士苹果     | 水果       | 京东生鲜    | 约1kg       | 四川京东生鲜有限公司     | 2023-06-10 |
| G00003   | 黑龙江有机大米     | 粮油       | 五常稻花香  | 5kg         | 黑龙江五常稻花香有限公司 | 2023-05-20 |
| G00004   | 新疆特级大枣       | 坚果       | 西域香      | 500g        | 新疆西域香农产品有限公司 | 2023-04-25 |
| G00005   | 山东红心火龙果     | 水果       | 果鲜生      | 约2个       | 山东果鲜生有限公司       | 2023-08-01 |
| G00006   | 宁夏枸杞           | 保健品     | 百草味      | 250g        | 宁夏百草味有限公司       | 2023-03-18 |
| G00007   | 进口牛奶           | 乳制品     | 德运        | 1L          | 澳大利亚德运有限公司     | 2023-07-05 |
| G00008   | 西湖龙井茶         | 茶叶       | 龙井村      | 250g        | 浙江龙井村有限公司       | 2023-04-30 |
| G00009   | 东北野生松茸       | 菌菇       | 东北老林    | 500g        | 黑龙江东北老林有限公司   | 2023-06-12 |
| G00010   | 广西沃柑           | 水果       | 新希望      | 约5个       | 广西新希望有限公司       | 2023-05-22 |
| G00011   | 安溪铁观音         | 茶叶       | 八马茶业    | 250g        | 福建八马茶业有限公司     | 2023-05-15 |
| G00012   | 青海牦牛肉         | 肉类       | 三江源      | 500g        | 青海三江源有限公司       | 2023-06-28 |
| G00013   | 内蒙古风干牛肉     | 肉类       | 蒙牛        | 250g        | 内蒙古蒙牛有限公司       | 2023-03-30 |
| G00014   | 海南椰子           | 水果       | 椰岛        | 3个         | 海南椰岛有限公司         | 2023-07-18 |
| G00015   | 贵州茅台酒         | 酒水       | 茅台        | 500ml       | 贵州茅台酒厂             | 2023-01-15 |
| G00016   | 进口橄榄油         | 油类       | 贝蒂斯      | 1L          | 西班牙贝蒂斯有限公司     | 2023-02-22 |
| G00017   | 国产蜂蜜           | 调味品     | 百花        | 500g        | 北京百花蜂业有限公司     | 2023-04-01 |
| G00018   | 有机鸡蛋           | 蛋类       | 农夫山泉    | 30个        | 浙江农夫山泉有限公司     | 2023-05-28 |
| G00019   | 新疆库尔勒香梨     | 水果       | 果园        | 约6个       | 新疆果园有限公司         | 2023-07-11 |
| G00020   | 进口红酒           | 酒水       | 拉菲        | 750ml       | 法国拉菲酒庄             | 2023-03-05 |
| G00021   | 苹果 iPhone 14     | 手机       | Apple       | 1台         | Apple Inc.               | 2023-01-10 |
| G00022   | 小米 12            | 手机       | 小米        | 1台         | 小米科技有限公司         | 2023-02-15 |
| G00023   | 华为 P50           | 手机       | 华为        | 1台         | 华为技术有限公司         | 2023-03-20 |
| G00024   | 三星 Galaxy S22    | 手机       | 三星        | 1台         | 三星电子有限公司         | 2023-04-25 |
| G00025   | OPPO Find X5       | 手机       | OPPO        | 1台         | OPPO电子有限公司         | 2023-05-30 |
| G00026   | 索尼 Bravia 电视   | 电器       | 索尼        | 1台         | 索尼公司                 | 2023-01-05 |
| G00027   | 飞利浦 电动剃须刀  | 电器       | 飞利浦      | 1台         | 飞利浦公司               | 2023-02-20 |
| G00028   | 戴森 吸尘器        | 电器       | 戴森        | 1台         | 戴森公司                 | 2023-03-15 |
| G00029   | 美的 空气净化器    | 电器       | 美的        | 1台         | 美的集团                 | 2023-04-10 |
| G00030   | 海尔 冰箱          | 电器       | 海尔        | 1台         | 海尔集团                 | 2023-05-05 |
| G00031   | 大连海参           | 海鲜       | 大连渔业    | 500g        | 大连渔业有限公司         | 2023-01-25 |
| G00032   | 青岛对虾           | 海鲜       | 青岛渔业    | 1kg         | 青岛渔业有限公司         | 2023-02-12 |
| G00033   | 舟山带鱼           | 海鲜       | 舟山渔业    | 1kg         | 舟山渔业有限公司         | 2023-03-05 |
| G00034   | 福建鲍鱼           | 海鲜       | 福建渔业    | 500g        | 福建渔业有限公司         | 2023-04-18 |
| G00035   | 海南龙虾           | 海鲜       | 海南渔业    | 1kg         | 海南渔业有限公司         | 2023-05-10 |
| G00036   | 山西陈醋           | 调味品     | 老陈醋      | 500ml       | 山西老陈醋有限公司       | 2023-01-01 |
| G00037   | 宁夏葡萄酒         | 酒水       | 宁夏红      | 750ml       | 宁夏红葡萄酒有限公司     | 2023-02-02 |
| G00038   | 江苏大米           | 粮油       | 苏北        | 5kg         | 江苏苏北大米有限公司     | 2023-03-01 |
| G00039   | 吉林木耳           | 菌菇       | 长白山      | 250g        | 吉林长白山有限公司       | 2023-04-04 |
| G00040   | 上海青菜           | 蔬菜       | 上海菜农    | 500g        | 上海菜农有限公司         | 2023-05-05 |
| G00041   | 福建柚子           | 水果       | 柚香园      | 1个         | 福建柚香园有限公司       | 2023-06-06 |
| G00042   | 贵州绿茶           | 茶叶       | 黔茶        | 250g        | 贵州黔茶有限公司         | 2023-07-07 |
| G00043   | 湖南腊肉           | 肉类       | 湘味        | 500g        | 湖南湘味有限公司         | 2023-08-08 |
| G00044   | 浙江乌龙茶         | 茶叶       | 浙茶        | 250g        | 浙江浙茶有限公司         | 2023-09-09 |
| G00045   | 安徽黄山毛峰       | 茶叶       | 徽茶        | 250g        | 安徽徽茶有限公司         | 2023-10-10 |
| G00046   | 湖北莲藕           | 蔬菜       | 鄂菜        | 500g        | 湖北鄂菜有限公司         | 2023-11-11 |
| G00047   | 广东荔枝           | 水果       | 岭南        | 1kg         | 广东岭南果品有限公司     | 2023-12-12 |
| G00048   | 陕西猕猴桃         | 水果       | 秦岭        | 1kg         | 陕西秦岭果品有限公司     | 2023-01-13 |
| G00049   | 云南普洱茶         | 茶叶       | 滇茶        | 250g        | 云南滇茶有限公司         | 2023-02-14 |
| G00050   | 四川青花椒         | 调味品     | 川味        | 200g        | 四川川味有限公司         | 2023-03-15 |
+----------+--------------------+------------+-------------+-------------+--------------------------+------------+
50 rows in set (0.01 sec)
```

对于销售记录，我们共插入了 200 多条记录，这里由于后续操作都是和 2024 年有关系，所以这里的日期都是 2024 年。这里故意有些日期超过了当前日期，为的是检验后续的命令确实达到了要求。这里我们把销售单价设成了随机，实现了每个商店的同一商品价格可能不同，价格和现实生活可能有较大出入，但满足本数据库的要求。

```
mysql> INSERT INTO SalesOrders (sale_id, goods_id, client_id, sale_price, store_id, sale_num, sale_sumprice, sale_date)
    -> VALUES
    -> ('SO0001', 'G00026', '0007', 74.38, '0000005', 3, 223.14, '2024-10-29'),
    -> ('SO0002', 'G00024', '0003', 54.58, '0000004', 1, 54.58, '2024-03-16'),
    -> ('SO0003', 'G00043', '0001', 85.67, '0000005', 2, 171.34, '2024-06-16'),
    -> ('SO0004', 'G00027', '0024', 59.86, '0000010', 2, 119.72, '2024-05-17'),
    -> ('SO0005', 'G00026', '0031', 60.31, '0000006', 2, 120.62, '2024-05-06'),
    -> ('SO0006', 'G00015', '0002', 25.61, '0000008', 3, 76.83, '2024-05-30'),
    -> ('SO0007', 'G00027', '0012', 10.71, '0000011', 3, 32.13, '2024-09-15'),
    -> ('SO0008', 'G00012', '0007', 27.81, '0000010', 1, 27.81, '2024-06-26'),
    -> ('SO0009', 'G00038', '0027', 21.59, '0000013', 3, 64.77, '2024-06-20'),
    -> ('SO0010', 'G00021', '0008', 92.45, '0000008', 4, 369.8, '2024-07-30'),
    -> ('SO0011', 'G00001', '0018', 41.72, '0000005', 1, 41.72, '2024-01-15'),
    -> ('SO0012', 'G00044', '0003', 10.42, '0000001', 4, 41.68, '2024-08-22'),
    -> ('SO0013', 'G00002', '0005', 58.31, '0000006', 2, 116.62, '2024-09-29'),
    -> ('SO0014', 'G00003', '0009', 48.21, '0000012', 1, 48.21, '2024-12-10'),
    -> ('SO0015', 'G00020', '0024', 58.87, '0000007', 5, 294.35, '2024-10-17'),
    -> ('SO0016', 'G00022', '0003', 46.46, '0000009', 3, 139.38, '2024-04-29'),
    -> ('SO0017', 'G00024', '0004', 88.18, '0000005', 5, 440.9, '2024-08-19'),
    -> ('SO0018', 'G00044', '0017', 45.23, '0000013', 5, 226.15, '2024-08-10'),
```

SO0179	G00030	0002	18.80	0000007	4	75.20	2024-12-21 00:00:00
SO0180	G00025	0018	66.78	0000002	3	200.34	2024-12-26 00:00:00
SO0181	G00042	0022	64.50	0000004	1	64.50	2024-10-12 00:00:00
SO0182	G00031	0007	77.27	0000001	5	386.35	2024-11-03 00:00:00
SO0183	G00005	0016	78.27	0000001	1	78.27	2024-07-20 00:00:00
SO0184	G00022	0021	64.64	0000013	3	193.92	2024-10-22 00:00:00
SO0185	G00014	0029	48.58	0000010	3	145.74	2024-06-18 00:00:00
SO0186	G00023	0025	77.11	0000003	4	308.44	2024-12-28 00:00:00
SO0187	G00015	0011	74.14	0000006	2	148.28	2024-01-05 00:00:00
SO0188	G00039	0034	22.73	0000011	4	90.92	2024-06-04 00:00:00
SO0189	G00020	0013	15.50	0000005	2	31.00	2024-04-24 00:00:00
SO0190	G00007	0014	32.63	0000007	1	32.63	2024-04-26 00:00:00
SO0191	G00027	0005	18.43	0000009	1	18.43	2024-11-05 00:00:00
SO0192	G00047	0031	43.69	0000004	5	218.45	2024-09-16 00:00:00
SO0193	G00028	0034	40.00	0000014	5	200.00	2024-04-10 00:00:00
SO0194	G00018	0032	66.41	0000012	4	265.64	2024-12-12 00:00:00
SO0195	G00004	0012	17.40	0000005	4	69.60	2024-12-23 00:00:00
SO0196	G00013	0006	13.56	0000001	2	27.12	2024-09-21 00:00:00
SO0197	G00036	0003	44.63	0000004	2	89.26	2024-06-06 00:00:00
SO0198	G00035	0018	30.22	0000011	4	120.88	2024-06-14 00:00:00
SO0199	G00010	0034	30.77	0000001	3	92.31	2024-11-12 00:00:00
SO0200	G00046	0023	37.78	0000015	4	151.12	2024-04-25 00:00:00
SO0201	G00026	0007	74.38	0000005	3	223.14	2024-10-29 00:00:00
SO0202	G00026	0008	74.38	0000005	3	223.14	2024-10-29 00:00:00
SO0203	G00026	0011	72.38	0000005	3	217.14	2024-10-29 00:00:00
SO0204	G00026	0014	72.38	0000005	3	217.14	2024-10-29 00:00:00
SO0205	G00026	0019	72.38	0000003	3	217.14	2024-10-29 00:00:00
SO0206	G00026	0012	72.38	0000003	3	217.14	2024-10-29 00:00:00
SO0207	G00026	0011	72.38	0000003	3	217.14	2024-10-29 00:00:00
SO0208	G00026	0021	72.38	0000003	3	217.14	2024-10-29 00:00:00
SO0209	G00026	0023	72.38	0000003	3	217.14	2024-10-29 00:00:00
SO0210	G00026	0028	72.38	0000003	3	217.14	2024-10-29 00:00:00
SO0250	G00011	0001	15.00	0000001	5	75.00	2024-01-02 00:00:00

对于 Ordering，我们共插入了 100 条左右的记录，同样的，实现了同一商品在不同商店的进货价格可能不同，不同时期可能也不同。

```
mysql> INSERT INTO Ordering (ordering_id, goods_id, store_id, ordering_price, ordering_num, ordering_sum, ordering_date)

    -> values
    -> ('OR00001', 'G00029', '0000012', 69.37, 4, 277.48, '2024-05-24'),
    -> ('OR00002', 'G00001', '0000002', 84.42, 3, 253.26, '2024-07-05'),
    -> ('OR00003', 'G00044', '0000014', 14.26, 1, 14.26, '2024-02-11'),
    -> ('OR00004', 'G00016', '0000001', 47.79, 3, 143.37, '2024-06-28'),
    -> ('OR00005', 'G00040', '0000004', 90.51, 5, 452.55, '2024-05-20'),
    -> ('OR00006', 'G00006', '0000007', 37.21, 5, 186.05, '2024-08-20'),
    -> ('OR00007', 'G00031', '0000003', 89.36, 2, 178.72, '2024-01-17'),
    -> ('OR00008', 'G00035', '0000003', 47.17, 1, 47.17, '2024-04-18'),
    -> ('OR00009', 'G00016', '0000001', 15.39, 4, 61.56, '2024-05-28'),
    -> ('OR00010', 'G00039', '0000011', 67.09, 1, 67.09, '2024-11-05'),
    -> ('OR00011', 'G00020', '0000001', 45.09, 3, 135.27, '2024-11-22'),
    -> ('OR00012', 'G00039', '0000015', 79.76, 4, 319.04, '2024-01-28'),
    -> ('OR00013', 'G00037', '0000005', 18.82, 2, 37.64, '2024-03-11'),
    -> ('OR00014', 'G00034', '0000001', 39.74, 1, 39.74, '2024-10-01'),
    -> ('OR00015', 'G00022', '0000013', 11.32, 4, 45.28, '2024-04-12'),
    -> ('OR00016', 'G00012', '0000013', 57.7, 5, 288.5, '2024-08-09'),
    -> ('OR00017', 'G00006', '0000004', 73.79, 5, 368.95, '2024-03-15'),
    -> ('OR00018', 'G00044', '0000010', 18.24, 4, 72.96, '2024-08-21'),
```

```
| OR00088     | G00040   | 0000003  |          58.26 |           4 |       233.04 | 2024-06-17 00:00:00 |
| OR00089     | G00039   | 0000004  |          53.08 |           3 |       159.24 | 2024-03-17 00:00:00 |
| OR00090     | G00009   | 0000010  |          23.41 |           2 |        46.82 | 2024-04-27 00:00:00 |
| OR00091     | G00001   | 0000007  |          57.22 |           2 |       114.44 | 2024-02-12 00:00:00 |
| OR00092     | G00019   | 0000009  |          54.98 |           2 |       109.96 | 2024-02-21 00:00:00 |
| OR00093     | G00012   | 0000007  |          70.68 |           3 |       212.04 | 2024-11-10 00:00:00 |
| OR00094     | G00025   | 0000013  |          56.27 |           5 |       281.35 | 2024-11-16 00:00:00 |
| OR00095     | G00001   | 0000014  |          30.53 |           3 |        91.59 | 2024-05-02 00:00:00 |
| OR00096     | G00023   | 0000008  |          58.76 |           3 |       176.28 | 2024-08-22 00:00:00 |
| OR00097     | G00022   | 0000005  |          59.76 |           5 |       298.80 | 2024-08-13 00:00:00 |
| OR00098     | G00043   | 0000004  |          49.43 |           5 |       247.15 | 2024-01-10 00:00:00 |
| OR00099     | G00049   | 0000003  |          36.66 |           2 |        73.32 | 2024-11-16 00:00:00 |
| OR00100     | G00004   | 0000005  |          35.32 |           5 |       176.60 | 2024-03-07 00:00:00 |
| OR0051      | G00011   | 0000001  |          15.00 |          10 |       150.00 | 2024-01-02 00:00:00 |
+-------------+----------+----------+----------------+-------------+--------------+---------------------+
```

Stock 表，我们获取不到真实数据，而且每个商店的每个商品几乎都有数据，所以这里用自动生成数据的方法生成了 750 条数据，保证每个商店的每个商品都有数据。

```
mysql> -- 填充 Stock 表
mysql> INSERT INTO Stock (store_id, goods_id, stock_num)
    -> SELECT s.store_id, g.goods_id, FLOOR(RAND() * 100) + 1  -- 随机生成 1 到 100 之间的库存数量
    -> FROM goods g
    -> CROSS JOIN (
    ->     SELECT store_id
    ->     FROM store
    ->     ORDER BY RAND()
    ->     LIMIT 15
    -> ) s;
Query OK, 750 rows affected (0.03 sec)
Records: 750  Duplicates: 0  Warnings: 0
```

5.4 查询操作

- 每个商店最畅销（购买客户最多的）的商品 ID 和客户数量?

这里可能会有一些并列第一的情况

```
mysql> WITH order_counts AS (
    ->     SELECT
    ->         so.store_id,
    ->         so.goods_id,
    ->         COUNT(*) AS order_count,
    ->         RANK() OVER (PARTITION BY so.store_id ORDER BY COUNT(*) DESC) AS rank_order_count
    ->     FROM
    ->         SalesOrders so
    ->     GROUP BY
    ->         so.store_id, so.goods_id
    -> )
    ->
    -> SELECT
    ->     s.store_id,
    ->     s.store_name,
    ->     oc.goods_id AS best_selling_goods_id,
    ->     oc.order_count AS max_order_count,
    ->     COUNT(DISTINCT so.client_id) AS customer_count
    -> FROM
    ->     order_counts oc
    -> JOIN
    ->     store s ON oc.store_id = s.store_id
    -> JOIN
    ->     SalesOrders so ON oc.store_id = so.store_id AND oc.goods_id = so.goods_id
    -> WHERE
    ->     oc.rank_order_count = 1  -- 只选择销售订单数量排名第一的商品
    -> GROUP BY
    ->     s.store_id, s.store_name, oc.goods_id, oc.order_count
    -> ORDER BY
    ->     s.store_id, oc.order_count DESC;
```

```
+----------+-------------------------+----------------------+-----------------+----------------+
| store_id | store_name              | best_selling_goods_id | max_order_count | customer_count |
+----------+-------------------------+----------------------+-----------------+----------------+
| 0000001  | 京东七鲜超市(王府井店)   | G00042               |               2 |              2 |
| 0000001  | 京东七鲜超市(王府井店)   | G00044               |               2 |              2 |
| 0000002  | 京东之家(西单店)         | G00010               |               2 |              2 |
| 0000002  | 京东之家(西单店)         | G00018               |               2 |              2 |
| 0000003  | 京东电器(中关村店)       | G00026               |               6 |              6 |
| 0000004  | 京东便利店(国贸店)       | G00036               |               3 |              3 |
| 0000005  | 京东生鲜(朝阳大悦城店)   | G00026               |               5 |              4 |
| 0000006  | 京东七鲜超市(华强北店)   | G00002               |               2 |              2 |
| 0000007  | 京东之家(南山店)         | G00007               |               3 |              3 |
| 0000008  | 京东电器(宝安店)         | G00009               |               1 |              1 |
| 0000008  | 京东电器(宝安店)         | G00010               |               1 |              1 |
| 0000008  | 京东电器(宝安店)         | G00015               |               1 |              1 |
| 0000008  | 京东电器(宝安店)         | G00021               |               1 |              1 |
| 0000008  | 京东电器(宝安店)         | G00030               |               1 |              1 |
| 0000008  | 京东电器(宝安店)         | G00039               |               1 |              1 |
| 0000008  | 京东电器(宝安店)         | G00044               |               1 |              1 |
| 0000008  | 京东电器(宝安店)         | G00045               |               1 |              1 |
| 0000008  | 京东电器(宝安店)         | G00049               |               1 |              1 |
| 0000009  | 京东便利店(罗湖店)       | G00016               |               2 |              2 |
| 0000009  | 京东便利店(罗湖店)       | G00022               |               2 |              2 |
| 0000010  | 京东生鲜(龙华店)         | G00012               |               1 |              1 |
| 0000010  | 京东生鲜(龙华店)         | G00014               |               1 |              1 |
| 0000010  | 京东生鲜(龙华店)         | G00027               |               1 |              1 |
| 0000010  | 京东生鲜(龙华店)         | G00031               |               1 |              1 |
| 0000010  | 京东生鲜(龙华店)         | G00043               |               1 |              1 |
| 0000011  | 京东七鲜超市(中央大街店)  | G00027               |               2 |              2 |
| 0000012  | 京东之家(哈西店)         | G00041               |               2 |              2 |
| 0000013  | 京东电器(江北店)         | G00032               |               2 |              2 |
| 0000014  | 京东便利店(学府路店)     | G00024               |               2 |              2 |
| 0000015  | 京东生鲜(道外店)         | G00046               |               3 |              3 |
+----------+-------------------------+----------------------+-----------------+----------------+
30 rows in set (0.00 sec)
```

- 除了牛奶之外，顾客购买最多的 3 种商品是什么?

这里只需把牛奶排除掉，进行排名即可。

```
mysql> SELECT
    ->     goods_id,
    ->     SUM(sale_num) AS total_sale_num
    -> FROM
    ->     SalesOrders
    -> WHERE
    ->     goods_id != 'G00007'  -- 排除牛奶商品
    -> GROUP BY
    ->     goods_id
    -> ORDER BY
    ->     total_sale_num DESC
    -> LIMIT 3;
+----------+----------------+
| goods_id | total_sale_num |
+----------+----------------+
| G00026   |             40 |
| G00022   |             24 |
| G00004   |             23 |
+----------+----------------+
3 rows in set (0.00 sec)
```

• 每个市（地址）最畅销的手机（类型）的前两种品牌（品牌）及销售总数;

这里防止选了一些空品牌的手机进入，所以在 where 里加入了排除空值的品牌和排除空字符串的品牌。

```
mysql> WITH ranked_brands AS (
    ->     SELECT
    ->         s.city,
    ->         g.goods_type AS phone_type,
    ->         g.goods_brand AS phone_brand,
    ->         SUM(so.sale_num) AS total_sales,
    ->         ROW_NUMBER() OVER(PARTITION BY s.city, g.goods_type ORDER BY SUM(so.sale_num) DESC) AS brand_rank
    ->     FROM
    ->         SalesOrders so
    ->     JOIN
    ->         goods g ON so.goods_id = g.goods_id
    ->     JOIN
    ->         store s ON so.store_id = s.store_id
    ->     WHERE
    ->         g.goods_type = '手机'  -- 筛选手机类型的商品
    ->         AND g.goods_brand IS NOT NULL  -- 排除空值的品牌
    ->         AND g.goods_brand <> ''       -- 排除空字符串的品牌
    ->     GROUP BY
    ->         s.city, g.goods_type, g.goods_brand
    -> )
    -> SELECT
    ->     city,
    ->     phone_type,
    ->     phone_brand,
    ->     total_sales
    -> FROM
    ->     ranked_brands
    -> WHERE
    ->     brand_rank <= 2  -- 选择每个城市每种手机类型的前两种品牌
    -> ORDER BY
    ->     city, phone_type, total_sales DESC;
+--------+------------+-------------+-------------+
| city   | phone_type | phone_brand | total_sales |
+--------+------------+-------------+-------------+
| 北京   | 手机       | 小米        |           9 |
| 北京   | 手机       | OPPO        |           8 |
| 哈尔滨 | 手机       | 三星        |           7 |
| 哈尔滨 | 手机       | OPPO        |           5 |
| 深圳   | 手机       | 小米        |          12 |
| 深圳   | 手机       | Apple       |           6 |
+--------+------------+-------------+-------------+
6 rows in set (0.00 sec)
```

• 查询商店（ID=‘02’），今年迄今为止前三个月销售商品 ID=‘1001’ 销售总额；

这里由于没有 ID= ‘1001’的商品，所以我改为了查询 ID=‘G00010’的商品。

```
mysql> SELECT SUM(sale_sumprice) AS total_sales
    ->  FROM SalesOrders
    -> WHERE store_id = '0000002' AND goods_id = 'G00010' AND sale_date >= DATE_SUB(CURDATE(), INTERVAL 3 MONTH)
    -> AND sale_date <= CURDATE();
+-------------+
| total_sales |
+-------------+
|      482.20 |
+-------------+
1 row in set (0.00 sec)
```

5.5 函数与过程

- 编写过程：给定一个城市，查询该市的商店 ID 及名称；

```
mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE GetStoresByCity(
    ->     IN city_name VARCHAR(50)
    -> )
    -> BEGIN
    ->     SELECT store_id, store_name
    ->     FROM store
    ->     WHERE city = city_name;
    -> END$$
ERROR 1304 (42000): PROCEDURE GetStoresByCity already exists
mysql>
mysql> DELIMITER ;
mysql> DELIMITER $$
mysql> CREATE PROCEDURE GetStoreByCity( IN city_name VARCHAR(50) )
    -> BEGIN
    -> SELECT store_id, store_name FROM store
    -> WHERE city = city_name;
    -> END$$
Query OK, 0 rows affected (0.00 sec)

mysql>  DELIMITER ;
mysql> CALL GetStoreByCity('北京');
+----------+------------------------+
| store_id | store_name             |
+----------+------------------------+
| 0000001  | 京东七鲜超市(王府井店) |
| 0000002  | 京东之家(西单店)       |
| 0000003  | 京东电器(中关村店)     |
| 0000004  | 京东便利店(国贸店)     |
| 0000005  | 京东生鲜(朝阳大悦城店) |
+----------+------------------------+
5 rows in set (0.00 sec)
```

- 编写函数：给定商店 ID，统计该店上个季度的销售总数；

```
mysql> CREATE FUNCTION SalesTotalLastQuarter(
    ->     store_id CHAR(7)
    -> )
    -> RETURNS DECIMAL(10,2)
    -> READS SQL DATA
    -> BEGIN
    ->     DECLARE start_date DATE;
    ->     DECLARE end_date DATE;
    ->     DECLARE total_sales DECIMAL(10,2);
    ->
    ->     -- 获取上个季度的开始日期和结束日期
    ->     SET start_date = DATE_SUB(DATE_FORMAT(NOW(), '%Y-%m-01'), INTERVAL 3 MONTH);
    ->     SET end_date = LAST_DAY(DATE_SUB(DATE_FORMAT(NOW(), '%Y-%m-01'), INTERVAL 1 MONTH));
    ->
    ->     -- 计算销售总额
    ->     SELECT SUM(sale_sumprice) INTO total_sales
    ->     FROM SalesOrders
    ->     WHERE store_id = store_id
    ->     AND sale_date >= start_date
    ->     AND sale_date <= end_date;
    ->
    ->     RETURN COALESCE(total_sales, 0);  -- 如果没有销售记录，返回0
    -> END$$
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT SalesTotalLastQuarter('0000001');
+-----------------------------------+
| SalesTotalLastQuarter('0000001') |
+-----------------------------------+
|                           7228.90 |
+-----------------------------------+
1 row in set (0.01 sec)
```

5.6 事件和触发器

- 编写事件：为每个客户生成上个月的账单（如每月 1 日 0 点）；

这里先创建一个存储客户账单的表，然后编写事件，最后还展示了一些事件。

```
mysql> CREATE TABLE MonthlyBills (
    ->     bill_id INT AUTO_INCREMENT NOT NULL,
    ->     client_id CHAR(4) NOT NULL,
    ->     month_start_date DATE NOT NULL,
    ->     month_end_date DATE NOT NULL,
    ->     total_amount DECIMAL(10,2) NOT NULL,
    ->     PRIMARY KEY (bill_id),
    ->     FOREIGN KEY (client_id) REFERENCES Client(client_id)
    ->         ON DELETE CASCADE ON UPDATE CASCADE,
    ->     CHECK (total_amount >= 0)
    -> );
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> DELIMITER $$
mysql>
mysql> CREATE EVENT GenerateMonthlyBill
    -> ON SCHEDULE
    ->     EVERY 1 MONTH
    ->     STARTS (CURRENT_DATE() + INTERVAL 1 DAY)  -- 每月1日0点开始执行
    -> DO
    -> BEGIN
    ->     DECLARE current_month_start DATE;
    ->     DECLARE current_month_end DATE;
    ->
    ->     -- 计算上个月的开始日期和结束日期
    ->     SET current_month_start = DATE_SUB(DATE_FORMAT(NOW(), '%Y-%m-01'), INTERVAL 1 MONTH);
    ->     SET current_month_end = LAST_DAY(DATE_SUB(DATE_FORMAT(NOW(), '%Y-%m-01'), INTERVAL 1 MONTH));
    ->
    ->     -- 为每个客户生成上个月的账单
    ->     INSERT INTO MonthlyBills (client_id, month_start_date, month_end_date, total_amount)
    ->     SELECT
    ->         client_id,
    ->         current_month_start,
    ->         current_month_end,
    ->         SUM(sale_sumprice) AS total_amount
    ->     FROM SalesOrders
    ->     WHERE sale_date >= current_month_start AND sale_date <= current_month_end
    ->     GROUP BY client_id;
    ->
    ->     -- 这里假设MonthlyBills是一个表，用于存储客户的月度账单数据
    -> END$$
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
```

```
mysql> show create  EVENT GenerateMonthlyBill;
+---------------------+----------------------------------------------------------------------------------------------------------+----------+------------------------------
| Event               | sql_mode                                                                                                 | time_zone | Create Event



          | character_set_client | collation_connection | Database Collation |
+---------------------+----------------------------------------------------------------------------------------------------------+----------+------------------------------
| GenerateMonthlyBill | ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION | SYSTEM    | CREATE DEFINER=`root`@`localho
st` EVENT `GenerateMonthlyBill` ON SCHEDULE EVERY 1 MONTH STARTS '2024-06-22 00:00:00' ON COMPLETION NOT PRESERVE ENABLE DO BEGIN
    DECLARE current_month_start DATE;
    DECLARE current_month_end DATE;


    SET current_month_start = DATE_SUB(DATE_FORMAT(NOW(), '%Y-%m-01'), INTERVAL 1 MONTH);
    SET current_month_end = LAST_DAY(DATE_SUB(DATE_FORMAT(NOW(), '%Y-%m-01'), INTERVAL 1 MONTH));


    INSERT INTO MonthlyBills (client_id, month_start_date, month_end_date, total_amount)
    SELECT
        client_id,
        current_month_start,
        current_month_end,
        SUM(sale_sumprice) AS total_amount
    FROM SalesOrders
    WHERE sale_date >= current_month_start AND sale_date <= current_month_end
    GROUP BY client_id;


END | gbk                  | gbk_chinese_ci       | utf8mb4_0900_ai_ci |
+---------------------+----------------------------------------------------------------------------------------------------------+----------+------------------------------
```

创建更新触发器，限制商品销售价格，一次上调不能超过 10%。

还做了个示例，进行试验，看是否创建成功。

```
mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER limit_price_increase
    -> BEFORE UPDATE ON SalesOrders
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.sale_price > OLD.sale_price * 1.10 THEN
    ->         SIGNAL SQLSTATE '45000'
    ->         SET MESSAGE_TEXT = 'Error: Price increase cannot exceed 10%';
    ->     END IF;
    -> END//
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> -- 示例：尝试将销售订单价格增加超过10%
mysql> UPDATE SalesOrders
    -> SET sale_price = sale_price * 1.15
    -> WHERE sale_id = 'SO0001';
ERROR 1644 (45000): Error: Price increase cannot exceed 10%
mysql>
```

模拟销售过程数据库变化，并编写相关触发器；

```
mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER after_sales_insert
    -> AFTER INSERT ON SalesOrders
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE current_stock INT;
    ->
    ->     -- 获取当前库存量
    ->     SELECT stock_num INTO current_stock
    ->     FROM Stock
    ->     WHERE store_id = NEW.store_id AND goods_id = NEW.goods_id;
    ->
    ->     -- 检查库存是否足够
    ->     IF current_stock < NEW.sale_num THEN
    ->         SIGNAL SQLSTATE '45000'
    ->         SET MESSAGE_TEXT = 'Error: Not enough stock available';
    ->     ELSE
    ->         -- 更新库存量
    ->         UPDATE Stock
    ->         SET stock_num = stock_num - NEW.sale_num
    ->         WHERE store_id = NEW.store_id AND goods_id = NEW.goods_id;
    ->     END IF;
    -> END//
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> select * from Stock
    -> where goods_id ='G00011' and store_id ='0000001';
+----------+----------+-----------+
| store_id | goods_id | stock_num |
+----------+----------+-----------+
| 0000001  | G00011   |        36 |
+----------+----------+-----------+
1 row in set (0.00 sec)
```

```
mysql> INSERT INTO SalesOrders (sale_id, goods_id, client_id, sale_price, store_id, sale_num, sale_sumprice, sale_date)
    -> VALUES ('SO00251', 'G00011', '0001', 15.00, '0000001', 5, 75.00, '2024-01-02');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select * from Stock
    -> where goods_id ='G00011' and store_id ='0000001';
+----------+----------+-----------+
| store_id | goods_id | stock_num |
+----------+----------+-----------+
| 0000001  | G00011   |        31 |
+----------+----------+-----------+
1 row in set (0.00 sec)
```

模拟订货过程数据库变化，并编写相关触发器；

```
mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER after_ordering_insert
    -> AFTER INSERT ON Ordering
    -> FOR EACH ROW
    -> BEGIN
    ->     -- 检查是否存在库存记录
    ->     IF EXISTS (SELECT 1 FROM Stock WHERE store_id = NEW.store_id AND goods_id = NEW.goods_id) THEN
    ->         -- 更新库存量
    ->         UPDATE Stock
    ->         SET stock_num = stock_num + NEW.ordering_num
    ->         WHERE store_id = NEW.store_id AND goods_id = NEW.goods_id;
    ->     ELSE
    ->         -- 如果没有现有库存记录，则插入新的库存记录
    ->         INSERT INTO Stock (store_id, goods_id, stock_num)
    ->         VALUES (NEW.store_id, NEW.goods_id, NEW.ordering_num);
    ->     END IF;
    -> END//
Query OK, 0 rows affected (0.02 sec)
```

```
mysql>
mysql> DELIMITER ;
mysql> select * from Ordering
    -> where goods_id = 'G00011' and store_id = '0000001';
Empty set (0.01 sec)

mysql> select * from Stock
    -> where goods_id = 'G00011' and store_id = '0000001';
+----------+----------+-----------+
| store_id | goods_id | stock_num |
+----------+----------+-----------+
| 0000001  | G00011   |        31 |
+----------+----------+-----------+
1 row in set (0.00 sec)
```

```
mysql> INSERT INTO Ordering (ordering_id, goods_id, store_id, ordering_price, ordering_num, ordering_sum, ordering_date)
    -> VALUES ('OR0051', 'G00011', '0000001', 15.00, 10, 150.00, '2024-01-02');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from Stock
    -> where goods_id = 'G00011' and store_id = '0000001';
+----------+----------+-----------+
| store_id | goods_id | stock_num |
+----------+----------+-----------+
| 0000001  | G00011   |        41 |
+----------+----------+-----------+
1 row in set (0.00 sec)
```

编写了一个触发器，当库存量低于一个阈值时，自动生成一个订货申请

这里订货申请肯定是与直接订货是不一样的，因为最终绝对到底要不要订货肯定是商店去再次确认的，所以这里专门生成一个订货申请的表。

```
mysql> CREATE TABLE ReorderRequests (
    ->     request_id CHAR(10) NOT NULL,
    ->     store_id CHAR(7) NOT NULL,
    ->     goods_id CHAR(10) NOT NULL,
    ->     request_date DATE NOT NULL,
    ->     status VARCHAR(20) DEFAULT 'Pending',
    ->     PRIMARY KEY (request_id),
    ->     FOREIGN KEY (store_id) REFERENCES Store(store_id)
    ->         ON DELETE CASCADE ON UPDATE CASCADE,
    ->     FOREIGN KEY (goods_id) REFERENCES Goods(goods_id)
    ->         ON DELETE CASCADE ON UPDATE CASCADE
    -> );
Query OK, 0 rows affected (0.10 sec)
```

```
mysql> DELIMITER $$
mysql> CREATE TRIGGER GenerateReorderRequest
    -> AFTER INSERT ON SalesOrders
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE current_stock INT;
    ->     DECLARE reorder_level INT;
    ->
    ->     -- 获取当前商品的库存量和订货阈值
    ->     SELECT stock_num INTO current_stock
    ->     FROM Stock
    ->     WHERE store_id = NEW.store_id AND goods_id = NEW.goods_id;
    ->
    ->     SELECT reorder_level INTO reorder_level
    ->     FROM Goods
    ->     WHERE goods_id = NEW.goods_id;
    ->
    ->     -- 检查库存量是否少于订货阈值，如果是则生成订货申请
    ->     IF current_stock < reorder_level THEN
    ->         INSERT INTO ReorderRequests (request_id, store_id, goods_id, request_date)
    ->         VALUES (
    ->             CONCAT('REQ', LPAD(FLOOR(RAND() * 10000), 4, '0')),
    ->             NEW.store_id,
    ->             NEW.goods_id,
    ->             CURDATE()
    ->         );
    ->     END IF;
    -> END$$
Query OK, 0 rows affected (0.01 sec)
```

6 总结

在本次设计数据库过程中，我遇到了很多困难。比如在设计表之间的关联关系时，遇到了如何设置外键以及确保数据一致性的困难。上网又仔细学习外键约束和参照完整性约束的使用，才成功解决了这个问题。还有在编写一些复杂的查询语句时，遇到了多表连接和子查询的挑战。通过不断学习和实践，掌握了 JOIN 操作、子查询和聚合函数的使用，成功编写出满足需求的复杂查询语句。这次设计中，我认为最难的是生成测试数据的过程，遇到了如何生成合理且符合实际业务逻辑的数据的困难。我通过编写脚本和 python，上网搜索真实数据，调教 GTP 和使用随机数生成器，才成功生成了满足需求的测试数据，确保了数据库的有效性。在设计 E-R 图时，也是一直很犹豫，怕设计错误，因为一旦整体结构错误，后面的都要推倒重来。反反复复设计了三次，才最终设计出了最终的 E-R 图整体结构。这次数据库设计，真真正正地算是把理论课学到的知识运用到实际上操作，加深了对知识的升华理解，并且感受到了设计数据库的真心不容易。最后，真心感谢老师和助教师兄师姐们辛苦阅读我的报告，辛苦啦!