データベース期末レポート

桑原 周平

ソーシャル・データサイエンス学部 2 年
5123017X

2024 年 11 月 4 日

1　データベース化対象

1.1　用いるデータの概要

今回、データベースの作成に際し元にするデータセットとして、Kaggle の小売データ「Online Retail Data Set」を活用することにする。このデータセットは、イギリスを拠点とするオンライン小売業者の実際の取引記録が 2010 年から 2011 年にかけて収集されたものである。541009 にもわたる取引データが集積されており、以下の 8 項目から成る。

- InvoiceNo (請求書番号): 注文を一意に識別するための番号
- StockCode (商品コード): 各商品の一意のコード
- Description (商品説明): 商品の簡潔な説明
- Quantity (数量): その取引で販売された数量
- InvoiceDate (注文日時): 取引が行われた日時
- UnitPrice (単価): 商品 1 単位の価格
- CustomerID (顧客 ID): 顧客を一意に識別する番号
- Country (国): 顧客の所在地

1.2　選定理由及び活用例

実際のオンライン小売業者の取引データを用いることで、実務で活用できるようなデータベースの作成が行えることを期待し、このデータセットを選定した。購買記録のデータを扱うことにした理由は、私が専攻したいと考えているマーケティングで有用だと考えたからである。第一に、地域ごとの特定の商品の購入頻度の違いを調べることにより、国ごとの嗜好を分析することが可能だ。第二に、商品ごとの売上や利益率を把握することで、在庫管理の最適化に活かすことができる。このように、小売業者が利潤を最大化するために、過去の取引情報にアクセスし、戦略を考えるための基盤となるようなデータベースを作成することが本稿の目的である。

2　データベースの設計

2.1　要求仕様 (改善前)

1. エンティティ
   ”顧客”がある時点に”商品”を”発注”し、その記録が行われていくことから、まずは、「顧客」、「商品」、「発注」の三者をエンティティとして設定する。
2. 属性
   - 顧客の属性: Country、CustomerID
   - 発注の属性: InvoiceNo、InvoiceDate、Quantity
   - 商品の属性: Unitprice、Description、StockCode

3. リレーション
   - 顧客が発注を”行う”。
     →濃度: 1 対多 (一人の顧客は複数の発注を行うことができる。)
     →オプショナリティ: 必須/任意 (発注には必ず一人の顧客が必要だが、顧客は発注をしない場合もある。)
   - 商品が発注の”対象となる”。
     →濃度: 多対 1 (一つの商品が別の発注において登場することも可能だが、上述の問題点の指摘にもあるように、一つの発注には 1 種類の商品しか含むことができない (要改善)。
     →オプショナリティ: 必須/任意 (発注が行われるには、とある商品が存在しなくてはならないが、ある商品に着目して、それが発注されなくても問題はない。)

2.2 ER 図 (改善前)

前述の要求仕様を再現した ER 図は、以下のようになる。(Google スライドにより作成)

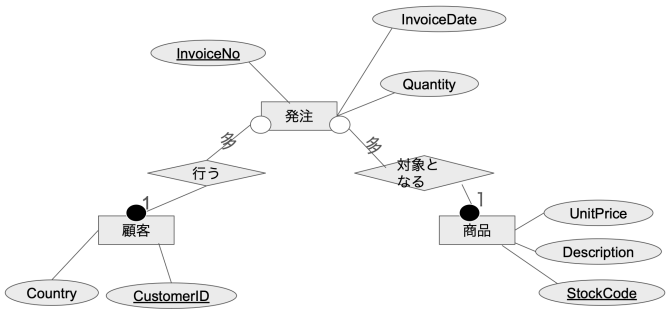

図 1　ER 図 (改善前)

2.3 上記の設計の問題点

発注エンティティが商品情報を含むことで、一つの発注に 1 つの商品しか含まれない前提となっており、複数の種類の商品の同時購入を扱えない。これは、別の種類の商品の購入を別の発注として扱うことになり、同時に購入されがちな補完財の把握等ができなくなってしまう。この解決策は、正規化の際に検討することとする。

3　データベースの構築

3.1　前節の設計に基づくリレーションスキーマ

前節の設計に基づくリレーションスキーマは、以下のようになる。

1. 顧客 (Customer)
   Customer(CustomerID, Country)
   主キー: CustomerID
2. 商品 (Product)
   Product(StockCode, Description, UnitPrice)
   主キー: StockCode
3. 発注 (Order)
   Order(InvoiceNo, InvoiceDate, Quantity)
   主キー: InvoiceNo

しかし、Order リレーションの主キーが InvoiceNo のみであると、一つの発注に一つの商品しか含めることができない。よって、発注詳細 (OrderDetail) という新たなエンティティを導入することにより解決し、得られたリレーションスキーマを正規化する。

3.2　修正後の ER 図

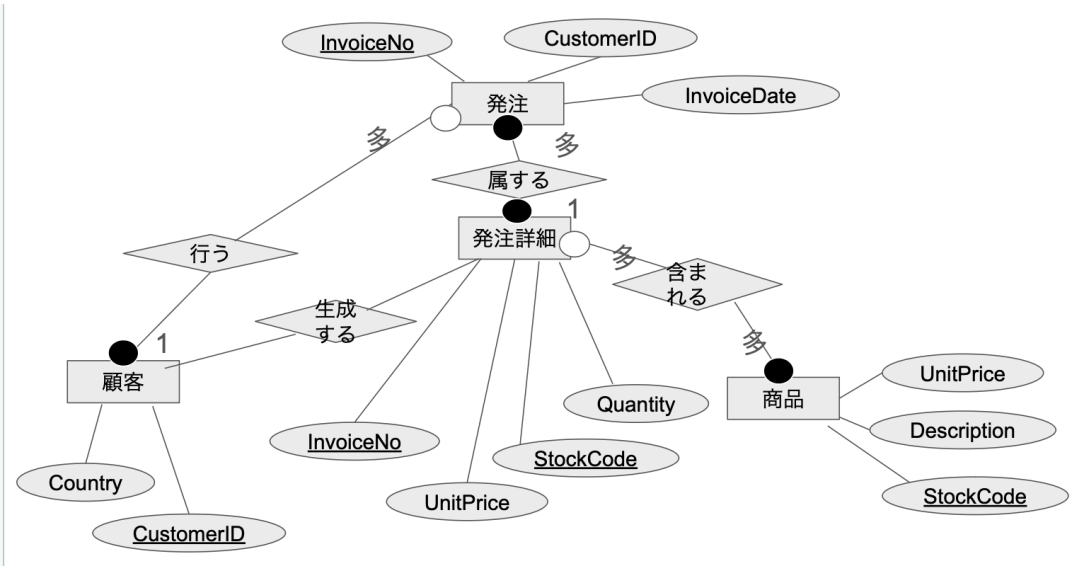

図 2　ER 図 (改善後)

3.3 修正後のリレーションスキーマ

1. 顧客 (Customer)
   Customer(CustomerID, Country)
   主キー: CustomerID
2. 商品 (Product)
   Product(StockCode, Description, Unitprice)
   主キー: StockCode
3. 発注 (Order)
   Order(Invoice, InvoiceDate, CustomerID)
   主キー: InvoiceNo
   外部キー: CustomerID (顧客エンティティを参照)
4. 発注詳細 (OrderDetail)
   OrderDetail(InvoiceNo, UnitPrice, StockCode, Quantity)
   主キー: (InvoiceNo, StockCode) (複合キー)
   外部キー: InvoiceNo(発注エンティティを参照)、StockCode(商品エンティティを参照)

3.4 修正後のリレーションスキーマの正規化

1. 第 1 正規形の確認
   各テーブルの属性が単一の値を持っており、第一正規形は満たされている。
2. 第 2 正規形の確認
   部分関数従属がないことを確認する。まず、Customer、Product、Order のテーブルでは、主キーが単一属性なので、部分関数従属性は存在しない。OrderDetail テーブルにおいて、UnitPrice は発注に依らず商品のみにより決まると考える場合、部分関数従属性が存在してしまうが、取引の時期や数量等の要因により、同じ商品でも異なる単価で取引されることがあり得るので、今回は両方の主キーに従属すると捉えることにする。よって、これは第 2 正規形も満たしている。
3. 第 3 正規形の確認
   推移的関数従属がないことを確認する。Customer テーブルでは、主キーの CustomerID に Country が従属しているだけなので、推移的関数従属は存在しない。Product テーブルでは、Description と UnitPrice が StockCode に従属しており、推移的従属はない。Order テーブルでは、InvoiceDate と CustomerID が InvoiceNo にのみ依存しており、推移的従属はない。OrderDetail テーブルでは、Quantity と Unitprice が主キーの複合キーのみに依存しているので、推移的従属は存在しない。よって、これはこれは第 3 正規形も満たしている。

3.5 SQLite でデータベースと表を作成

ソースコード 1　テーブル作成のコード

```
1
2
3  -- 顧客情報のテーブル
4  CREATE TABLE Customer (
5      CustomerID TEXT PRIMARY KEY,
6      Country TEXT
7  );
8
9  -- 商品情報のテーブル
10 CREATE TABLE Product (
11     StockCode TEXT PRIMARY KEY,
12     Description TEXT,
13     UnitPrice REAL
14 );
15
16 -- 発注情報のテーブル
17 CREATE TABLE Orders (
18     InvoiceNo TEXT PRIMARY KEY,
19     InvoiceDate TEXT,
20     CustomerID TEXT,
21     FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
22 );
23
24 -- 発注詳細のテーブル
25 CREATE TABLE OrderDetail (
26     InvoiceNo TEXT,
27     StockCode TEXT,
28     Quantity INTEGER,
29     UnitPrice REAL,
30     PRIMARY KEY (InvoiceNo, StockCode),
31     FOREIGN KEY (InvoiceNo) REFERENCES Orders(InvoiceNo),
32     FOREIGN KEY (StockCode) REFERENCES Product(StockCode)
33 );
```

4　データ収集

データベース化対象でも述べた通り、今回は既存のデータセットを用いるのでこちらは省略。

5　データベースへのデータ登録

5.1　テーブル構造を確認

ソースコード 2　期待通りのテーブルの構造となっているか確認

```
1
2  sqlite> .schema Customer
3  CREATE TABLE Customer (
4      CustomerID TEXT PRIMARY KEY,
5      Country TEXT
6  );
7  sqlite> .schema Product
8  CREATE TABLE Product (
9      StockCode TEXT PRIMARY KEY,
10     Description TEXT,
11     UnitPrice REAL
12 );
13 sqlite> .schema Orders
14 CREATE TABLE Orders (
15     InvoiceNo TEXT PRIMARY KEY,
16     InvoiceDate TEXT,
17     CustomerID TEXT,
18     FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
19 );
20 sqlite> .schema OrderDetail
21 CREATE TABLE OrderDetail (
22     InvoiceNo TEXT,
23     StockCode TEXT,
24     Quantity INTEGER,
25     UnitPrice REAL,
26     PRIMARY KEY (InvoiceNo, StockCode),
27     FOREIGN KEY (InvoiceNo) REFERENCES Orders(InvoiceNo),
28     FOREIGN KEY (StockCode) REFERENCES Product(StockCode)
29 );
```

5.2 csv ファイルの再編成とデータのインポート

テーブル構造に合わせて、data.csv のファイルを、product.csv、customer.csv、orders.csv、orderdetail.csv に分割する。(Google colaboratory 上で pandas を用いた。)

ソースコード 3　分割したデータのインポート (SQLite)

```
sqlite3 retail_data.db

sqlite> DELETE FROM Customer;
DELETE FROM Product;
DELETE FROM Orders;
DELETE FROM OrderDetail;

sqlite> .mode csv
.import '/Users/shuheikuwabara/Desktop/customer.csv' Customer
.import '/Users/shuheikuwabara/Desktop/product.csv' Product
.import '/Users/shuheikuwabara/Desktop/orders.csv' Orders
.import '/Users/shuheikuwabara/Desktop/orderdetail.csv' OrderDetail

sqlite> SELECT * FROM Customer LIMIT 5;
SELECT * FROM Product LIMIT 5;
SELECT * FROM Orders LIMIT 5;
SELECT * FROM OrderDetail LIMIT 5;

CustomerID,Country
17850.0,"United Kingdom"
13047.0,"United Kingdom"
12583.0,France
13748.0,"United Kingdom"

StockCode,Description,UnitPrice
85123A,"WHITE HANGING HEART T-LIGHT HOLDER",2.55
71053,"WHITE METAL LANTERN",3.39
84406B,"CREAM CUPID HEARTS COAT HANGER",2.75
84029G,"KNITTED UNION FLAG HOT WATER BOTTLE",3.39

InvoiceNo,InvoiceDate,CustomerID
536365,"2010/12/1 8:26",17850.0
536366,"2010/12/1 8:28",17850.0
536367,"2010/12/1 8:34",13047.0
```

```
536368,"2010/12/1 8:34",13047.0

InvoiceNo,StockCode,Quantity,UnitPrice
536365,85123A,6,2.55
536365,71053,6,3.39
536365,84406B,8,2.75
536365,84029G,6,3.39
```

5.3 データ型の確認と変更

データ型が適切であるか調べ、必要に応じて変更する。確認するコードは以下。

ソースコード 4　データ型の確認 (SQLite)

```
sqlite> .schema Customer
CREATE TABLE Customer (
    CustomerID TEXT PRIMARY KEY,
    Country TEXT
);
sqlite> .schema Product
CREATE TABLE Product (
    StockCode TEXT PRIMARY KEY,
    Description TEXT,
    UnitPrice REAL
);
sqlite> .schema Orders
CREATE TABLE Orders (
    InvoiceNo TEXT PRIMARY KEY,
    InvoiceDate TEXT,
    CustomerID TEXT,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);
sqlite> .schema OrderDetail
CREATE TABLE OrderDetail (
    InvoiceNo TEXT,
    StockCode TEXT,
    Quantity INTEGER,
    UnitPrice REAL,
    PRIMARY KEY (InvoiceNo, StockCode),
    FOREIGN KEY (InvoiceNo) REFERENCES Orders(InvoiceNo),
    FOREIGN KEY (StockCode) REFERENCES Product(StockCode)
```

```
28 );
```

CustomerID と InvoiceNo が TEXT 型になっているのが不適切なので、CustomerID と InvoiceNo を INTEGER 型に修正する。その結果が以下。

ソースコード 5　変更後のデータ型 (SQLite)

```
1  sqlite> .schema
2  CREATE TABLE Product (
3      StockCode TEXT PRIMARY KEY,
4      Description TEXT,
5      UnitPrice REAL
6  );
7  CREATE TABLE Customer (
8      CustomerID INTEGER PRIMARY KEY,
9      Country TEXT
10 );
11 CREATE TABLE IF NOT EXISTS "Orders" (
12     InvoiceNo INTEGER PRIMARY KEY,
13     InvoiceDate TEXT,
14     CustomerID INTEGER,
15     FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
16 );
17 CREATE TABLE IF NOT EXISTS "OrderDetail" (
18     InvoiceNo INTEGER,
19     StockCode TEXT,
20     Quantity INTEGER,
21     UnitPrice REAL,
22     PRIMARY KEY (InvoiceNo, StockCode),
23     FOREIGN KEY (InvoiceNo) REFERENCES Orders(InvoiceNo),
24     FOREIGN KEY (StockCode) REFERENCES Product(StockCode)
25 );
```

6 クエリ

6.1 クエリ１ (累計販売数トップ５の商品)

ソースコード 6 累計販売数トップ５の商品

```
SELECT Product.Description, SUM(OrderDetail.Quantity) AS TotalSold
FROM OrderDetail
JOIN Product ON OrderDetail.StockCode = Product.StockCode
GROUP BY Product.StockCode
ORDER BY TotalSold DESC
LIMIT 5;

SMALL POPCORN HOLDER|56249
WORLD WAR 2 GLIDERS ASSTD DESIGNS|53747
JUMBO BAG RED RETROSPOT|47191
WHITE HANGING HEART T-LIGHT HOLDER|38644
ASSORTED COLOUR BIRD ORNAMENT|35983
```

最も売れている上位５商品を抽出するようにした。これを見ることで、小売業者は人気商品を把握し、販売促進や在庫管理を効率的に行うことができる。また、流行を把握した上で潜在的に需要のある新製品を考案することも可能で、アーケティング戦略立案に有用となることが期待される。

6.2 クエリ２ (購入金額トップ５の商品)

ソースコード 7 購入金額トップ５の商品

```
SELECT Customer.CustomerID, Customer.Country, SUM(OrderDetail.Quantity * OrderDetail.Un
FROM OrderDetail
JOIN Orders ON OrderDetail.InvoiceNo = Orders.InvoiceNo
JOIN Customer ON Orders.CustomerID = Customer.CustomerID
GROUP BY Customer.CustomerID
ORDER BY TotalSpent DESC
LIMIT 5;

14646|Netherlands|280206.02
18102|United Kingdom|259657.3
17450|United Kingdom|194390.79
16446|United Kingdom|168472.5
14911|EIRE|143711.169999999
```

最も多くの金額分購入した顧客のトップ５ (出力が長くなりすぎないよう便宜上減らした) のユーザー ID と国名、総支出額を出力するようにした。これにより、重要顧客を特定し、積極的にパーソナライズされた広告を送信することができる。さらに、購買頻度の高いユーザーに対してクーポンを選択的に配布すること等も可能である。

6.3 クエリ３ (売上総額トップ５の国と総売上)

ソースコード 8 売上総額トップ５の国

```
1  SELECT
2      Customer.Country,
3      SUM(OrderDetail.Quantity) AS TotalQuantity
4  FROM
5      OrderDetail
6  JOIN
7      Orders ON OrderDetail.InvoiceNo = Orders.InvoiceNo
8  JOIN
9      Customer ON Orders.CustomerID = Customer.CustomerID
10 GROUP BY
11     Customer.Country
12 ORDER BY
13     TotalQuantity DESC
14 LIMIT 5;
15
16 United Kingdom|4231471
17 Netherlands|200937
18 EIRE|140379
19 Germany|119119
20 France|111301
```

売上総額を国ごとに集計し、最も売上が多い国から順に表示した結果、以上のようになった。これは、特定の国ごとの売上状況を把握し、売上が高い国に対して優先的にマーケティング戦略を立てること等を可能にする。