

Database Final Project Report

FOOD PRODUCTS

Team 3

NAME	NET ID
Chaoran Li	cxl190012
Miao Miao	mxm190020
Yinglu Huang	yxh190064

Project Description

In this project, we design a data system for FOOD PRODUCTS, including data requirements for food products, ER diagram, relational schema, normalization, create table by using SQL/PL

Data Requirement

In this project, we design a data system for FOOD PRODUCTS, including data requirements for food products, ER diagram, relational schema, normalization, create table by using SQL/PL.

1. Customer and Admin

website has two roles, admin and customer. They have several attributes in common, such as name, date of birth, username, and password, so we generalized a superclass People to hold these common features.

An admin should have an identification. As for Customers, there are payment methods, addresses, and contact to be its specific attributes.

2. Product

The product should have an ID as its key attribute. Also there are name, price, and product description as basic attributes to describe a product. We keep an expiration date to see how long could this product be kept in stock.

3. Batch

We use a batch system to manage products' incoming batch and its stock. For a batch, there should be

- a. batch ID: works as an identification;
- b. stock date: when this batch was put into storage;
- c. stock number: how much product there is in one batch;
- d. store number: how much product is still available in this batch.

4. Order

Once a customer places an order on the website, there should be a record of this order saved in the database.

An order is owned by a customer and related to products that the customer bought. A customer could have more than one order, also could have no order in the record.

Once a product is removed from the database, the corresponding item in an order should not be removed.

- a. order ID: works as an identification;
- b. purchase date: when the customer placed this order;
- c. comment: feedback from the customer who placed this order.

5. Shopping Cart

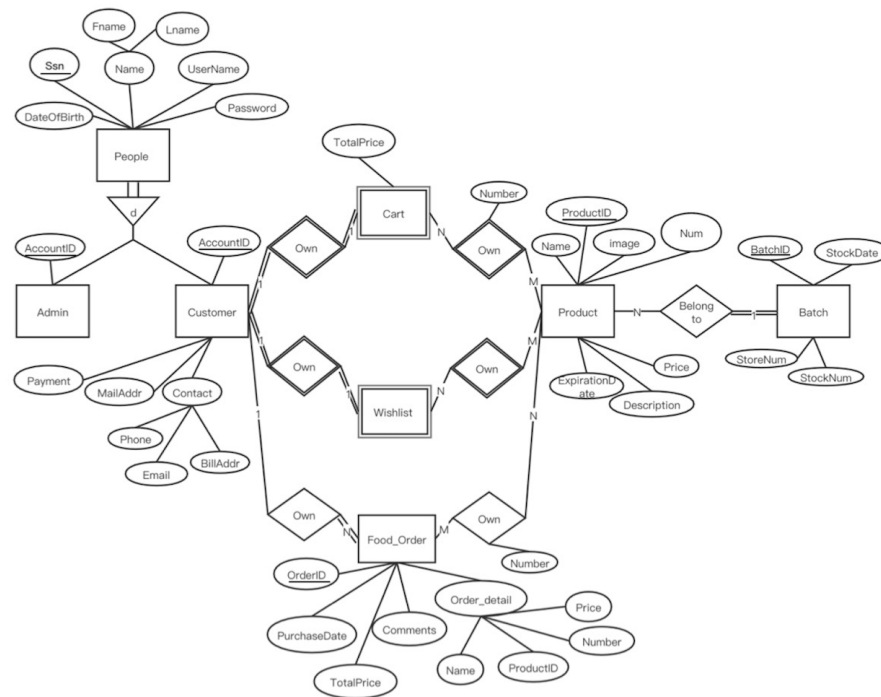
A shopping cart is owned by a customer, to save items that the customer wants to purchase later. A shopping cart could be empty.

Items in the cart should correspond to the products. There should be an amount of the product that the customer wants to purchase. Once a product is removed from the database, the item in a shopping cart should also be removed.

6. Wishlist

A customer should also have a wishlist, which is almost a shopping cart without the attribute of the product's amount.

ER Diagram



Relational Model

CUSTOMER (AccountID, Fname, Lname, DateOfBirth, SSN, UserName, Password, Payment, MailAddr, BillAddr, Phone, Email)

ADMIN (AccountID, Fname, Lname, DateOfBirth, SSN, UserName, Password)

BATCH (BatchID, ProductID[FK -> PRODUCT.ProductID], StoreNum, StockNum, StockDate)

PRODUCT (ProductID, Name, Price, Description, Image, ExpirationDate, Num)

FOOD_ORDER (OrderID, AccountID[FK -> CUSTOMER.AccountID], PurchaseDate, TotalPrice, Comments)

ORDERDETAIL (OrderID[FK -> ORDER.OrderID], Name, ProductID, Number, Price)

ORDER_OWN_PRODUCT (OrderID[FK -> ORDER.OrderID], ProductID[FK -> PRODUCT.ProductID], Number)

CART AccountID[FK -> CUSTOMER.AccountID], TotalPrice)

CART_OWN_PRODUCT (AccountID[FK -> CUSTOMER.AccountID], ProductID[FK -> PRODUCT.ProductID], Number)

WISHLIST (AccountID[FK -> CUSTOMER.AccountID])

WISHLIST_OWN_PRODUCT (AccountID[FK -> CUSTOMER.AccountID],
ProductID[FK -> PRODUCT.ProductID])

Normalization

There is no functional dependency that would violate 3NF, so the relational model we got from the ER model is already in 3NF.

Create tables

```
CREATE TABLE CUSTOMER (  
    AccountID    INT,  
    Fname        VARCHAR(25) NOT NULL,  
    Lname        VARCHAR(25) NOT NULL,  
    DateOfBirth  DATE,  
    SSN          CHAR(9),  
    UserName     VARCHAR(25) NOT NULL,  
    Password     VARCHAR(25) NOT NULL,  
    Payment      VARCHAR(20) NOT NULL,  
    MailAddr     CHAR(50) NOT NULL,  
    BillAddr     CHAR(50) NOT NULL,  
    Phone        CHAR(10),  
    Email        VARCHAR(40),  
    PRIMARY KEY (AccountID)  
);
```

```
CREATE TABLE ADMIN (  
    AccountID    INT,  
    Fname        VARCHAR(25) NOT NULL,  
    Lname        VARCHAR(25) NOT NULL,  
    DateOfBirth  DATE,  
    SSN          CHAR(9),  
    UserName     VARCHAR(25) NOT NULL,  
    Password     VARCHAR(25) NOT NULL,  
    PRIMARY KEY (AccountID)  
);
```

```
CREATE TABLE PRODUCT (  
    ProductID    INT,  
    Name         VARCHAR(50) NOT NULL,  
    Price        DECIMAL(10, 2) NOT NULL,  
    Description   VARCHAR(200),  
    Image        VARCHAR(50),  
    ExpirationDate INT NOT NULL,  
    Num          INT NOT NULL,  
    PRIMARY KEY (ProductID)  
);
```

```
CREATE TABLE BATCH (  
    BatchID      INT,  
    ProductID    INT NOT NULL,  
    StoreNum     INT NOT NULL,  
    StockNum     INT NOT NULL,  
    StockDate    DATE NOT NULL,  
    PRIMARY KEY (BatchID),  
    FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID) ON  
DELETE CASCADE  
);
```

```
CREATE TABLE FOOD_ORDER (  
    OrderID      INT,  
    AccountID    INT NOT NULL,  
    PurchaseDate DATE NOT NULL,  
    TotalPrice   DECIMAL(10, 2) NOT NULL,  
    Comments     VARCHAR(200),  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (AccountID) REFERENCES CUSTOMER(AccountID)  
ON DELETE CASCADE  
);
```

```
CREATE TABLE ORDER_OWN_PRODUCT (  
    OrderID      INT,  
    ProductID    INT,  
    Number INT NOT NULL,  
    PRIMARY KEY (OrderID, ProductID),  
    FOREIGN KEY (OrderID) REFERENCES FOOD_ORDER(OrderID) ON  
DELETE CASCADE,  
    FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID) ON  
DELETE CASCADE  
);
```

```
CREATE TABLE ORDER_DETAIL (  
    OrderID      INT,  
    ProductID    INT,  
    Name          VARCHAR(50) NOT NULL,  
    Price          DECIMAL(10, 2) NOT NULL,  
    Num            INT NOT NULL,  
    PRIMARY KEY (OrderID, ProductID),  
    FOREIGN KEY (OrderID) REFERENCES FOOD_ORDER(OrderID) ON  
DELETE CASCADE  
);
```

```
CREATE TABLE CART (  
    AccountID     INT NOT NULL,  
    TotalPrice    DECIMAL(10, 2) NOT NULL,  
    PRIMARY KEY (AccountID),  
    FOREIGN KEY (AccountID) REFERENCES CUSTOMER(AccountID)  
ON DELETE CASCADE  
);
```

```
CREATE TABLE CART_OWN_PRODUCT (  
    AccountID     INT,  
    ProductID     INT,  
    Num           INT NOT NULL,  
    PRIMARY KEY (AccountID, ProductID),  
    FOREIGN KEY (AccountID) REFERENCES CUSTOMER(AccountID)  
ON DELETE CASCADE,  
    FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID) ON  
DELETE CASCADE  
);
```

```
CREATE TABLE WISHLIST (
    AccountID INT NOT NULL,
    PRIMARY KEY (AccountID),
    FOREIGN KEY (AccountID) REFERENCES CUSTOMER(AccountID)
ON DELETE CASCADE
);
```

```
CREATE TABLE WISHLIST_OWN_PRODUCT (
    AccountID INT,
    ProductID INT,
    PRIMARY KEY (AccountID, ProductID),
    FOREIGN KEY (AccountID) REFERENCES CUSTOMER(AccountID)
ON DELETE CASCADE,
    FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID) ON
DELETE CASCADE
);
```

PL/SQL

Trigger

1. Update total price of a shopping cart

Add an attribute `TotalPrice` in class cart, to improve database performance. Under the premise of not violating 3NF, increase redundant data and improve efficiency.

```
--Trigger1 Update TotalPrice in CART after update
CART_OWN_PRODUCT
CREATE or REPLACE TRIGGER UpdateCartPriceForInsert
after INSERT on CART_OWN_PRODUCT
FOR EACH ROW
DECLARE
price          PRODUCT.Price%TYPE;
old_total_price CART.TotalPrice%TYPE;
total_price    CART.TotalPrice%TYPE;
BEGIN
    SELECT Price INTO price FROM PRODUCT WHERE
:NEW.ProductID = PRODUCT.ProductID;
    SELECT TotalPrice INTO old_total_price FROM CART WHERE
:NEW.AccountID = CART.AccountID;
```

```

        total_price := old_total_price + price * :NEW.Num;
        UPDATE CART SET TotalPrice = total_price WHERE
:NEW.AccountID = CART.AccountID;
        DBMS_OUTPUT.put_line('Updated customer ' ||
:NEW.AccountID || ''s cart total price from $' ||
old_total_price || ' to $' || total_price);
END;

CREATE or REPLACE TRIGGER UpdateCartPriceForUpdate
after UPDATE on CART_OWN_PRODUCT
FOR EACH ROW
DECLARE
price          PRODUCT.Price%TYPE;
old_total_price CART.TotalPrice%TYPE;
total_price    CART.TotalPrice%TYPE;
BEGIN
        SELECT Price INTO price FROM PRODUCT WHERE
:NEW.ProductID = PRODUCT.ProductID;
        SELECT TotalPrice INTO old_total_price FROM CART WHERE
:NEW.AccountID = CART.AccountID;
        total_price := old_total_price + price * (:NEW.Num -
:OLD.Num);
        UPDATE CART SET TotalPrice = total_price WHERE
:NEW.AccountID = CART.AccountID;
        DBMS_OUTPUT.put_line('Updated customer ' ||
:NEW.AccountID || ''s cart total price from $' ||
old_total_price || ' to $' || total_price);
END;

CREATE or REPLACE TRIGGER UpdateCartPriceForDelete
after DELETE on CART_OWN_PRODUCT
FOR EACH ROW
DECLARE
price          PRODUCT.Price%TYPE;
old_total_price CART.TotalPrice%TYPE;
total_price    CART.TotalPrice%TYPE;
BEGIN
        SELECT Price INTO price FROM PRODUCT WHERE
:OLD.ProductID = PRODUCT.ProductID;
        SELECT TotalPrice INTO old_total_price FROM CART WHERE
:OLD.AccountID = CART.AccountID;
        total_price := old_total_price - price * :OLD.Num;

```



```

        UPDATE CART SET TotalPrice = total_price WHERE
:OLD.AccountID = CART.AccountID;
        DBMS_OUTPUT.put_line('Updated customer ' ||
:OLD.AccountID || ''s cart total price from $' ||
old_total_price || ' to $' || total_price);
END;

```

2. Update total remaining quantity of the product

In class Product, add an attribute `Num` as the total remaining quantity of the product.

```

--Trigger2 Update Num in PRODUCT after update BATCH
CREATE or REPLACE TRIGGER UpdateProductNumForInsert
after INSERT on BATCH
FOR EACH ROW
DECLARE
old_num          PRODUCT.Num%TYPE;
new_num          PRODUCT.Num%TYPE;
BEGIN
    SELECT Num INTO old_num FROM PRODUCT WHERE
:NEW.ProductID = PRODUCT.ProductID;
    new_num := old_num + :NEW.StoreNum;
    UPDATE PRODUCT SET Num = new_num WHERE :NEW.ProductID =
PRODUCT.ProductID;
    DBMS_OUTPUT.put_line('Updated product ' ||
:NEW.ProductID || ''s number from ' || old_num || ' to ' ||
new_num);
END;

CREATE or REPLACE TRIGGER UpdateProductNumForUpdate
after Update on BATCH
FOR EACH ROW
DECLARE
old_num          PRODUCT.Num%TYPE;
new_num          PRODUCT.Num%TYPE;
BEGIN
    SELECT Num INTO old_num FROM PRODUCT WHERE
:NEW.ProductID = PRODUCT.ProductID;
    new_num := old_num + (:NEW.StoreNum - :OLD.StoreNum);
    UPDATE PRODUCT SET Num = new_num WHERE :NEW.ProductID =
PRODUCT.ProductID;

```

```

        DBMS_OUTPUT.put_line('Updated product ' ||
:NEW.ProductID || ''s number from ' || old_num || ' to ' ||
new_num);
END;

CREATE or REPLACE TRIGGER UpdateProductNumForDelete
after DELETE on BATCH
FOR EACH ROW
DECLARE
old_num          PRODUCT.Num%TYPE;
new_num          PRODUCT.Num%TYPE;
BEGIN
    SELECT Num INTO old_num FROM PRODUCT WHERE
:OLD.ProductID = PRODUCT.ProductID;
    new_num := old_num - :OLD.StoreNum;
    UPDATE PRODUCT SET Num = new_num WHERE :OLD.ProductID =
PRODUCT.ProductID;
    DBMS_OUTPUT.put_line('Updated product ' ||
:OLD.ProductID || ''s number from ' || old_num || ' to ' ||
new_num);
END;

```

Procedure

1. Retrieve the sum of price in a given date

Check by day, traverse order_own_product to check the sum of the total price of each order for the same day at the given time. Default date is system day.

```

--Procedure1 Daily summary
CREATE OR REPLACE PROCEDURE DailySummary (inpDate IN
FOOD_ORDER.PurchaseDate%TYPE DEFAULT SYSDATE) AS
    CURSOR order_cur IS SELECT * FROM FOOD_ORDER;
    order_row          FOOD_ORDER%ROWTYPE;
    daily_sum          FOOD_ORDER.TotalPrice%TYPE;
BEGIN
    daily_sum := 0;
    OPEN order_cur;
    LOOP
        FETCH order_cur INTO order_row;
        EXIT WHEN (order_cur%NOTFOUND);
    
```

```

        IF (inpDate = order_row.PurchaseDate) THEN
            daily_sum := daily_sum + order_row.TotalPrice;
            DBMS_OUTPUT.put_line('An order earns $' ||
order_row.TotalPrice);
        END IF;
    END LOOP;
    CLOSE order_cur;
    DBMS_OUTPUT.put_line('In ' || to_char(inpDate,
'mm/dd/yyyy') || ', we earned $' || daily_sum);
END;

```

2. Remove expired batch

Default input time is the system date. If a batch is not short sold and is removed, the second trigger will be triggered.

```

--Procedure2 Remove Expired Food Batch
CREATE OR REPLACE PROCEDURE RemoveExpiredFood(inpDate IN
FOOD_ORDER.PurchaseDate%TYPE DEFAULT SYSDATE) AS
    product_row          PRODUCT%ROWTYPE;
    difference            INTEGER;
    batch_row            BATCH%ROWTYPE;
    CURSOR batch_cur IS SELECT * FROM BATCH;
BEGIN
    OPEN batch_cur;
    LOOP
        FETCH batch_cur INTO batch_row;
        EXIT WHEN (batch_cur%NOTFOUND);
        SELECT * INTO product_row FROM PRODUCT WHERE
PRODUCT.ProductID = batch_row.ProductID;
        difference := ROUND(to_number(inpDate -
batch_row.StockDate), 0);
        IF (difference > product_row.ExpirationDate) THEN
            IF (batch_row.StoreNum > 0) THEN
                DBMS_OUTPUT.put_line('Remove ' ||
batch_row.StoreNum || ' expired ' || product_row.Name ||
'(s)');
            END IF;
            DELETE FROM BATCH WHERE BATCH.BatchID =
batch_row.BatchID;
        END IF;
    END LOOP;

```

```

CLOSE batch_cur;
DBMS_OUTPUT.put_line('Removed all expired food.');
```

END;

Achievement Exhibition

Trigger

1. Update total price of a shopping cart

The screenshot shows a SQL script in the 'Worksheet' tab of SQL Developer. The script performs the following operations:

- Line 41: `SELECT TotalPrice INTO old_total_price FROM CART WHERE :OLD.AccountID = CART.AccountID;`
- Line 42: `total_price := old_total_price - price * :OLD.Num;`
- Line 43: `UPDATE CART SET TotalPrice = total_price WHERE :OLD.AccountID = CART.AccountID;`
- Line 44: `DBMS_OUTPUT.put_line('Updated customer ' || :OLD.AccountID || ''s cart total price from $' || old_total_price || ' to $' || total_price);`
- Line 45: `END;`
- Line 46: (Empty line)
- Line 47: `INSERT INTO CART_OWN_PRODUCT (AccountID, ProductID, Num) VALUES (106, 2, 3);`
- Line 48: `UPDATE CART_OWN_PRODUCT SET Num = 5 WHERE (AccountID = 106 AND ProductID = 2);`
- Line 49: `DELETE FROM CART_OWN_PRODUCT WHERE (AccountID = 106 AND ProductID = 2);`
- Line 50: (Empty line)

The 'Script Output' tab shows the execution results:

- Task completed in 0.248 seconds
- Updated customer 106's cart total price from \$32.5 to \$41.5
- 1 row inserted.
- Updated customer 106's cart total price from \$41.5 to \$47.5
- 1 row updated.
- Updated customer 106's cart total price from \$47.5 to \$32.5
- 1 row deleted.

The screenshot shows a SQL script in the 'Worksheet' tab of SQL Developer. The script performs the following operations:

- Line 24: `24 BEGIN`
- Line 25: `SELECT Price INTO price FROM PRODUCT WHERE :NEW.ProductID = PRODUCT.ProductID;`
- Line 26: `SELECT TotalPrice INTO old_total_price FROM CART WHERE :NEW.AccountID = CART.AccountID;`
- Line 27: `total_price := old_total_price + price * (:NEW.Num - :OLD.Num);`
- Line 28: `UPDATE CART SET TotalPrice = total_price WHERE :NEW.AccountID = CART.AccountID;`
- Line 29: `DBMS_OUTPUT.put_line('Updated customer ' || :NEW.AccountID || ''s cart total price from $' || old_total_price || ' to $' || total_price);`
- Line 30: `END;`
- Line 31: (Empty line)
- Line 32: `UPDATE CART_OWN_PRODUCT SET Num = 5 WHERE (AccountID = 106 AND ProductID = 2);`
- Line 33: (Empty line)

The 'Script Output' tab shows the execution results:

- Task completed in 0.13 seconds
- Updated customer 106's cart total price from \$41.5 to \$47.5
- 1 row updated.

2. Update total remaining quantity of the product

Worksheet Query Builder

```

87 new_num := old_num - :OLD.StoreNum;
88 UPDATE PRODUCT SET Num = new_num WHERE :OLD.ProductID = PRODUCT.ProductID;
89 DBMS_OUTPUT.put_line('Updated product ' || :OLD.ProductID || ''s number from ' || old_num || ' to ' || new_num);
90 END;
91
92
93 INSERT INTO BATCH (BatchID, ProductID, StoreNum, StockNum, StockDate) VALUES (100, 3, 10, 10, to_date('04/25/2020','mm/dd/yyyy'));
94 UPDATE BATCH SET StoreNum = 5 WHERE BatchID = 100;
95 DELETE FROM BATCH WHERE BatchID = 100;
96

```

Script Output x

Task completed in 0.241 seconds

Updated product 3's number from 0 to 10

1 row inserted.

Updated product 3's number from 10 to 5

1 row updated.

Updated product 3's number from 5 to 0

1 row deleted.

Worksheet Query Builder

```

84 new_num
85 BEGIN
86     SELECT Num INTO old_num FROM PRODUCT WHERE :OLD.ProductID = PRODUCT.ProductID;
87     new_num := old_num - :OLD.StoreNum;
88     UPDATE PRODUCT SET Num = new_num WHERE :OLD.ProductID = PRODUCT.ProductID;
89     DBMS_OUTPUT.put_line('Updated product ' || :OLD.ProductID || ''s number from ' || old_num || ' to ' || new_num);
90 END;
91
92 DELETE FROM BATCH WHERE BatchID = 100;
93

```

Script Output x

Task completed in 0.094 seconds

Updated product 3's number from 10 to 0

1 row deleted.

Procedure

1. Retrieve the sum of price in a given date

Worksheet Query Builder

```

112 END IF;
113 END LOOP;
114 CLOSE order_cur;
115 DBMS_OUTPUT.put_line('In ' || to_char(inpDate, 'mm/dd/yyyy') || ', we earned $' || daily_sum);
116 END;
117
118 BEGIN
119 DailySummary (to_date('04/25/2020', 'mm/dd/yyyy'));
120 END;

```

Script Output x

Task completed in 0.144 seconds

An order earns \$33.77
 An order earns \$41.42
 An order earns \$18.16
 In 04/25/2020, we earned \$93.35

PL/SQL procedure successfully completed.

2. Remove expired batch

Worksheet Query Builder

```

148 INSERT INTO BATCH (batchID, ProductID, StoreNum, StockNum, StockDate)
149 VALUES (1, 0, 0, 5, to_date('04/25/2020', 'mm/dd/yyyy'));
150 INSERT INTO BATCH (BatchID, ProductID, StoreNum, StockNum, StockDate)
151 VALUES (2, 1, 5, 20, to_date('04/25/2020', 'mm/dd/yyyy'));
152 INSERT INTO BATCH (BatchID, ProductID, StoreNum, StockNum, StockDate)
153 VALUES (3, 2, 0, 5, to_date('04/24/2020', 'mm/dd/yyyy'));
154
155 BEGIN
156 RemoveExpiredFood (to_date('04/28/2020', 'mm/dd/yyyy'));
157 END;

```

Script Output x

Task completed in 0.128 seconds

Updated product 0's number from 0 to 0
 Updated product 0's number from 0 to 0
 Remove 5 expired meemaw doughnut(s)
 Updated product 1's number from 5 to 0
 Updated product 2's number from 1 to 1
 Removed all expired food.

PL/SQL procedure successfully completed.

Worksheet Query Builder

```

154
155 BEGIN
156 RemoveExpiredFood (to_date('04/28/2020', 'mm/dd/yyyy'));
157 END;

```

Script Output x

Task completed in 0.133 seconds

Updated product 0's number from 0 to 0

1 row inserted.

Updated product 0's number from 0 to 0

1 row inserted.

Updated product 1's number from 0 to 5

1 row inserted.

Updated product 2's number from 1 to 1

1 row inserted.

Updated product 0's number from 0 to 0
 Updated product 0's number from 0 to 0
 Remove 5 expired meemaw doughnut(s)
 Updated product 1's number from 5 to 0
 Updated product 2's number from 1 to 1
 Removed all expired food.

PL/SQL procedure successfully completed.