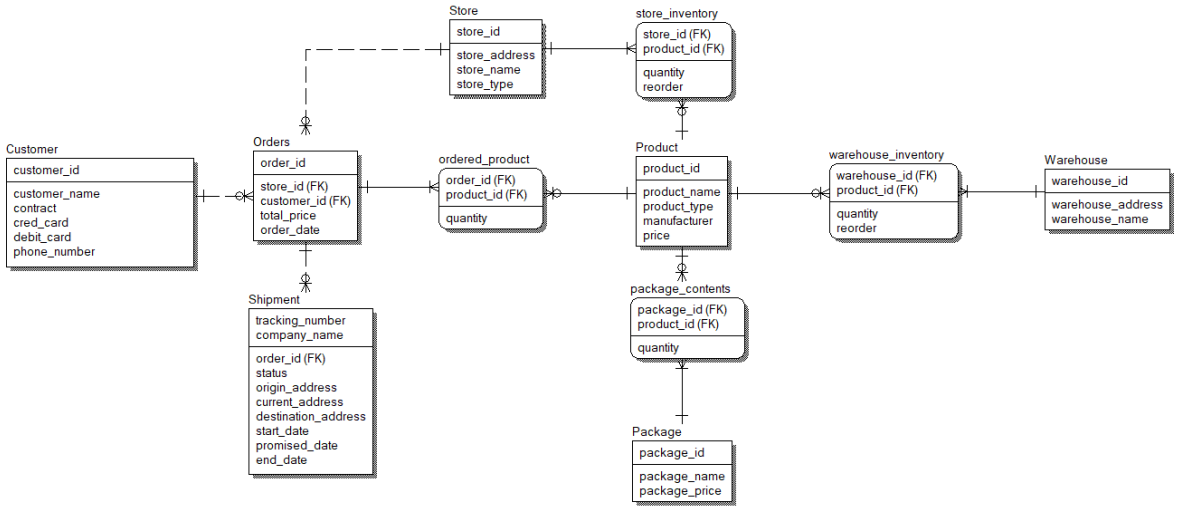
CSE4110 – Database System

Project 2. Normalization and Query Processing

20171645 박찬우

1. Decomposition to BCNF



우선, 후에 서술할 Query를 고려하여 기존의 logical schema에서 몇가지 변경점이 추가되었다. Customer Table에 고객의 연락처를 저장하는 phone_number attribute를 추가하였고, package_contents 에 특정 package에 어떤 종류의 물건이 몇 개 들어있는지를 나타내는 quantity attribute를 추가하였다. 추가로, 기존의 Order table의 이름을 Orders로 변경했는데, 이는 query중 order가 query 명령어중 하나여서 table명으로 인식이 안되기 때문이다.

위 내용은 모두 기존 Schema에서 Query를 위해 필요한 정보를 추가한 것들이고, 기존 Relational Schema에서 BCNF form으로 decompose 해야 하는데, 기존 Relational Schema의 모든 Table이 BCNF form이므로, 추가적인 decomposition 과정은 없다.

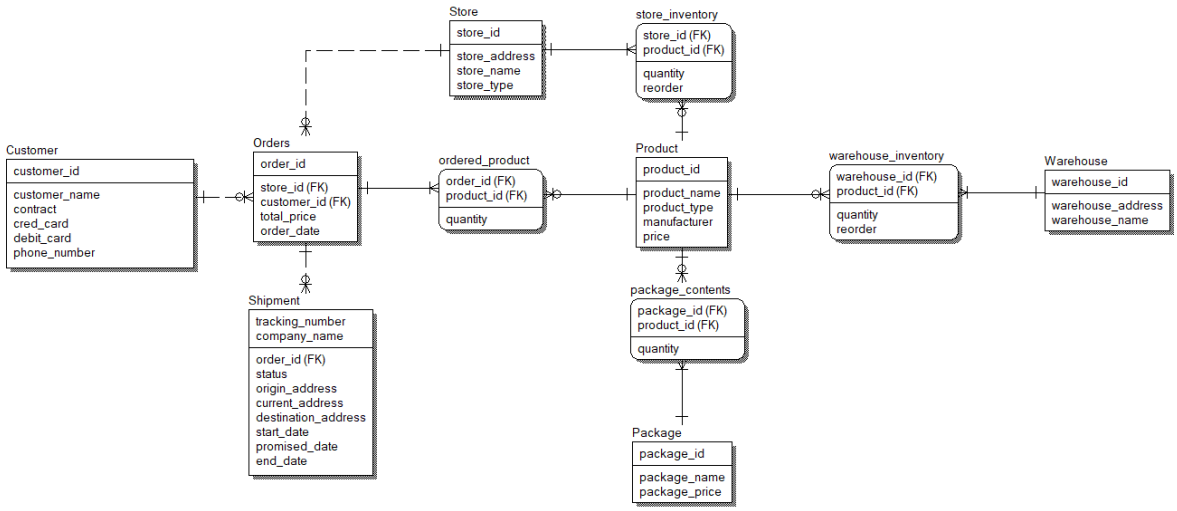
기존의 Relational Schema의 Functional dependency는 모두 a->b 에 대하여 a가 Relational Schema의 super key인 경우, 예를들어 [customer_id -> customer_name, contract, cred_card, debit_card, phone_number] 인 경우와 trivial 한 경우, 예를 들어 [customer_name , contract] -> [customer_name, contract] 와 같은 두 가지 종류를 제외한 functional dependency가 존재하지 않는다. 따라서, 전부 BCNF form 형태를 취하고 있으므로, 별다른 decomposition 과정이 없다.

간단히 Customer relation에 대해 BCNF simplified test를 해보자면, 현재 Customer에 존재

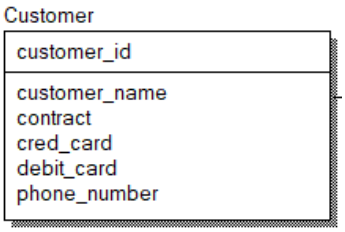
하는 functional dependency는 trivial한 functional dependency를 제외하면 [customer_id -> customer_name, contract, cred_card, debit_card, phone_number] 하나이다. 동명이인이 있을 수 있고, contract 여부는 binary하며 cred_card, debit_card, phone_number는 nullable하므로 해당 attribute로 functional dependency를 형성할 수 없다. 따라서 trivial한 functional dependency를 제외한 위의 하나의 functional dependency의 경우, a->b 형태에서 a가 relation의 primary key 이므로, super key이다. 따라서, Customer는 BCNF form을 만족한다.

다른 모든 Relation 역시 비슷하게 trivial 및 a가 super key인 functional dependency를 제외한 다른 functional dependency가 없으므로, 전부 BCNF form을 만족한다.

2. Physical Schema



각 Entity의 attribute에 대해 먼저 살펴본다.



customer_id 는 각 고객을 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

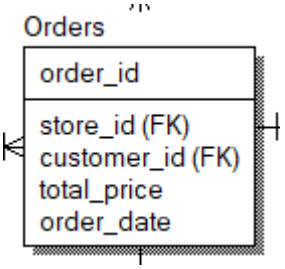
contract는 고객이 회사와 계약을 했는지 여부를 나타내는 attribute로, boolean type의 데이터를 가지며 Null이 허용되지 않는다. contraint는 customer_contract로, boolean data인 yes와 no 값만을 가질수 있도록 제약한다.

cred_card는 고객의 신용카드 정보를 저장하는 attribute로, CHAR(19) type 데이터를 가지며 Null이 허용된다. card_form 이라는 contraint로 0000-0000-0000-0000 형태를 가진 값만 입력받을 수 있도록 제약한다.

debit_card는 고객의 체크카드 정보를 저장하는 attribute로, 이름을 제외하면 cred_card와 동일하다.

phone_number는 고객의 전화번호를 저장하는 attribute로, CHAR(13) type 데이터를 가지며 Null이 허용된다. phone_form 이라는 contraint로 000-0000-0000 형태의 값을 입력받을 수 있도록 제약한다.

이 Entity의 primary key는 customer_id 이다.



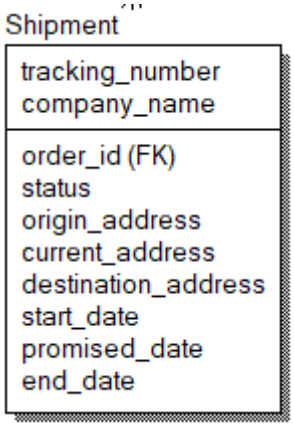
customer_id 는 각 주문을 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

store_id 와 customer_id는 각각 Store 와 Customer로부터 받아온 foreign key로, 둘 모두 INT type의 데이터를 가지며 Null이 허용되지 않는다.

total_price는 해당 주문의 총 금액($)을 저장하는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

order_date는 주문한 날짜 정보를 저장하는 attribute로, DATETIME type의 데이터를 가지며 Null이 허용되지 않는다. DATETIME type의 도메인은 Datetime으로, "2022-06-01 12:00:00" 와 같은 형태의 데이터를 저장할 수 있다.

해당 Entity의 primary key는 order_id이고, foreign key는 store_id, customer_id 이다.



tracking_number는 송장번호, 즉 배송 정보의 id 역할을 하는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

company_name은 배송하는 회사의 이름을 저장하는 attribute로, VARCHAR(100) type 데이터를 가져 최대 100자까지 문자열을 저장할 수 있고, Null이 허용되지 않는다.

order_id는 Orders로부터 받아온 foreign key로, INT type 데이터를 가지며 null이 허용되지 않는다.

status는 현재 배송이 어떤 상태인지를 나타내는 attribute로, VARCHAR(50) type 데이터를 가지며 Null이 허용되지 않는다. Shipment_Status 라는 constraint에 의해 "Information" "At Pickup" "Arrived at Carrier" "In transmit" "Out for Delivery" "Delivered" "Stopped by accident" 값들중 하나만 입력받을 수 있다. 해당 값들은 각각 '배송정보 접수', '화물 접수', '배송사 도착', '배송중', '배송직전', '배송완료', '문제 발생으로 배송중단' 을 의미한다.

origin_address, current_address, destination_address는 각각 배송 출발지, 현재 위치, 배송 목적지에 대한 정보를 저장하는 attribute로 모두 VARCHAR(100) type 데이터를 저장하며 Null이 허용되지 않는다.

start_date, promised_date, end_date는 각각 배송 시작 시간, 예정 도착 시간, 도착 시간에 대한 정보를 저장하는 attribute로 모두 DATETIME type 데이터를 저장하며, end_date만 Null이 허용되고, 나머지는 Null이 허용되지 않는다.

이 Entity의 primary key는 tracking_number와 company_name이고, foreign key는 order_id 이다.

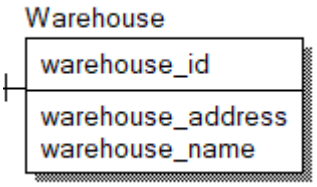
store_id 는 각 상점을 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

store_address는 상점의 주소를 저장하는 attribute로, VARCHAR(100) type의 데이터를 가지며 Null이 허용되지 않는다.

store_name은 상점의 이름을 저장하는 attribute로, VARCHAR(50) type의 데이터를 가지며 Null이 허용되지 않는다.

store_type은 상점이 온라인 상점인지 오프라인 상점인지를 나타내는 attribute로, VARCHAR(7) type 데이터를 저장하며 Null이 허용되지 않는다. Store_type constraint에 의해 online, offline 두가지 값만을 가질 수 있도록 제약된다.

이 Entity의 primary key는 store_id이다.

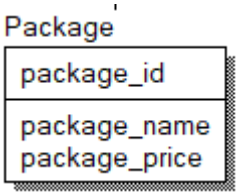


warehouse_id 는 각 창고를 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

warehouse_address는 창고의 주소를 저장하는 attribute로, VARCHAR(100) type의 데이터를 가지며 Null이 허용되지 않는다.

warehouse_name은 창고의 이름을 저장하는 attribute로, VARCHAR(50) type의 데이터를 가지며 Null이 허용되지 않는다.

이 Entity의 primary key는 warehouse_id이다.



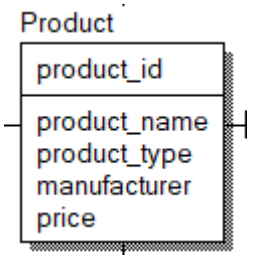
package_id 는 여러 상품의 묶음인 package를 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

package_name은 묶음의 이름을 나타내는 attribute로, VARCHAR(50) type 데이터를 가지며 Null이 허용되지 않는다.

package_price는 묶음의 가격을 타나내는 attribute로, INT type 데이터를 가지며 Null이

허용되지 않는다.

이 Entity의 primary key는 package_id이다.



product_id 는 상품을 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

product_name은 상품의 이름을 저장하는 attribute로, VARCHAR(50) type 데이터를 가지며 Null이 허용되지 않는다.

product_type은 상품의 종류를 저장하는 attribute로, VARCHAR(50) type 데이터를 가지며 Null이 허용되지 않는다.

manufacturer는 상품의 제조사를 저장하는 attribute로, VARCHAR(50) type 데이터를 가지며 Null이 허용되지 않는다.

price는 상품의 가격을 저장하는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

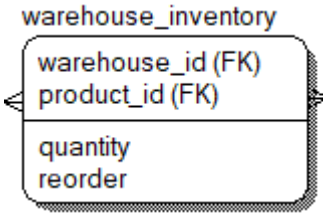
이 Entity의 primary key는 product_id이다.

store_id와 product_id는 각각 Store와 Product로부터 가져온 foreign key로, 모두 INT type 데이터를 가지며 Null을 허용하지 않는다.

quantity는 상점에 있는 물건의 개수를 저장하는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

reorder는 상점의 해당 상품의 재주문에 대한 정보를 저장하는 attribute로, VARCHAR(20) type 데이터를 가지며 Null이 허용되지 않는다. Reorder contraint로 인해 각 "Reorder need" "Reordering" "Not Need" 세가지 값만을 가지도록 제약된다. 각 값은 재주문이 필요함, 재주문중, 재주문이 끝났거나 필요치 않음을 의미한다.

이 Entity의 primary key는 store_id와 product_id이며, foreign_key 역시 store_id, product_id이다.

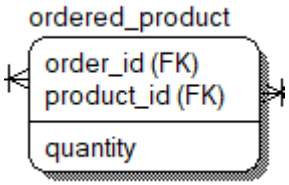


warehouse_id와 product_id는 각각 Warehouse와 Product로부터 가져온 foreign key로, 모두 INT type 데이터를 가지며 Null을 허용하지 않는다.

quantity는 창고에 있는 물건의 개수를 저장하는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

reorder는 창고의 해당 상품의 재주문에 대한 정보를 저장하는 attribute로, VARCHAR(20) type 데이터를 가지며 Null이 허용되지 않는다. Reorder contraint로 인해 각 "Reorder need" "Reordering" "Not Need" 세가지 값만을 가지도록 제약된다. 각 값은 재주문이 필요함, 재주문중, 재주문이 끝났거나 필요치 않음을 의미한다.

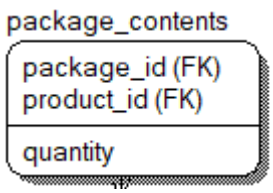
이 Entity의 primary key는 warehouse_id와 product_id이며, foreign_key 역시 warehouse_id, product_id이다.



order_id와 product_id는 각각 Orders와 Product로부터 가져온 foreign key로, 모두 INT type 데이터를 가지며 Null을 허용하지 않는다.

quantity는 해당 주문에서 어떤 상품을 몇 개 주문했는지를 의미하는 attribute로, INT type 데이터를 가지며 Null을 허용하지 않는다.

이 Entity의 primary key는 order_id, product_id이고 foreign key 역시 order_id, product_id 이다.



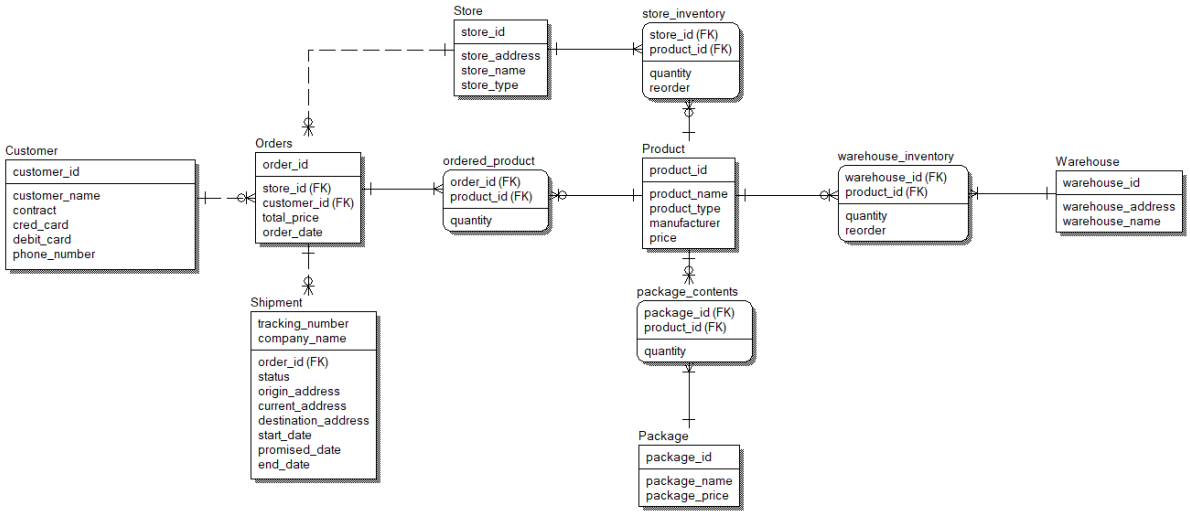
package_id와 product_id는 각각 Package와 Product로부터 가져온 foreign key로, 모두

INT type 데이터를 가지며 Null을 허용하지 않는다.

quantity는 해당 묶음에서 어떤 상품이 몇 개 포함됐는지를 의미하는 attribute로, INT type 데이터를 가지며 Null을 허용하지 않는다.

이 Entity의 primary key는 package_id, product_id이고 foreign key 역시 package_id, product_id이다.

이제 각 Entity 사이의 관계를 살펴본다.



Customer와 Orders 에서 order는 total participation해야 하며, Customer와 Orders 사이의 mapping cardinality는 one-to-many이다. 다시말해, customer는 order를 하나도 하지 않을 수 있고, 하나만 할 수 있으며, 여러 order를 할 수도 있다. 따라서, customer – order 사이의 relationship은 one-to-zero-one-or-more 로 설정한다. schema reduction rule에 따라, order는 customer의 primary key를 foreign key로 갖게 된다. 이때 customer_id 는 order의 primary key로 불필요하므로, 둘 사이의 relationship은 non-identifying relationship이다.

Orders와 Shipment 에서 shipment가 total participation해야 하며, order와 shipment 사이의 mapping cardinality는 one-to-many 이다. 다시말해, shipment는 하나의 order를 갖고, order는 shipment를 갖지 않을수도, 하나 이상을 가질수도 있다.

즉, order - shipment의 Relation은 one-to-zero-one-or-more 관계로 설정한다. 이에 따라 shipment는 order의 primary key를 foreign key로 갖는데, 그것이 order_id이다. 이때 order_id는 shipment의 primary_key로 불필요하므로 둘 사이의 relationship은 non-identifying relationship이다.

Orders와 Store 에서 order는 total participation해야 하며, order와 store 사이의

mapping cardinality는 many-to-one 이다. 즉, 한 매장이 많은 주문을 가질수 있지만, 한 주문이 많은 매장을 가질 순 없다. 한 매장은 주문이 없어도 되므로, store와 order 사이의 관계는 one-to-zero-one-or-more 관계를 이루게 된다. 이에 따라 order는 store의 primary key인 store_id를 foreign key로 받게 되는데, order의 primary key로 필요하지 않으므로 둘 사이의 관계는 non-identifying relationship이다.

Orders와 ordered_product, Product 에서 한번의 주문은 여러 제품을 포함할 수 있고, 아무 제품도 없을 수는 없으므로, order와 ordered_product 사이의 relationship은 one-to-one-or-more relationship이 성립한다. 한 제품이 여러 주문에 들어있을 수 있고, 아무 주문에도 안들어있어도 괜찮으므로 product와 ordered_product 사이의 relationship은 one-to-zero-one-or-many relationship이 성립한다. 두 relation 모두 ordered_product가 상대방으로부터 빌린 foreign key를 primary key로 가지므로, identifying relation이다.

Store와 store_inventory, Product 에서 한 상점에 여러 제품이 있을 수 있지만, 상점에서 아무 제품도 팔지 않을 수는 없으므로 store와 store_inventory 사이의 관계는 one-to-one-or-more relationship이 성립한다. 한 제품이 여러 상점에 있을 수 있고, 특정 제품은 아무 상점에서도 팔지 않을 수 있다. 따라서, product와 store_inventory 사이의 relationship은 one-to-zero-one-or-more relationship이 성립한다. 두 relation 모두 store_inventory가 상대방으로부터 빌린 foreign key를 primary key로 가지므로, identifying relation이다.

Warehouse와 warehouse_inventory, Product 에서 한 제품이 여러 창고에 있을 수 있고, 아무 창고에 없을수도 있다. 따라서, product와 warehouse_inventory는 one-to-zero-one-or-many relationship이 성립한다. 하지만, 한 창고에 여러 제품이 있을 수는 있지만, 창고에 아무 제품도 보관되지 않을 수는 없으므로 warehouse와 warehouse_inventory 사이의 relationship은 one-to-one-or-more relationship이 성립한다. 두 relation 모두 warehouse_inventory가 상대방으로부터 빌린 foreign key를 primary key로 가지므로, identifying relation이다.

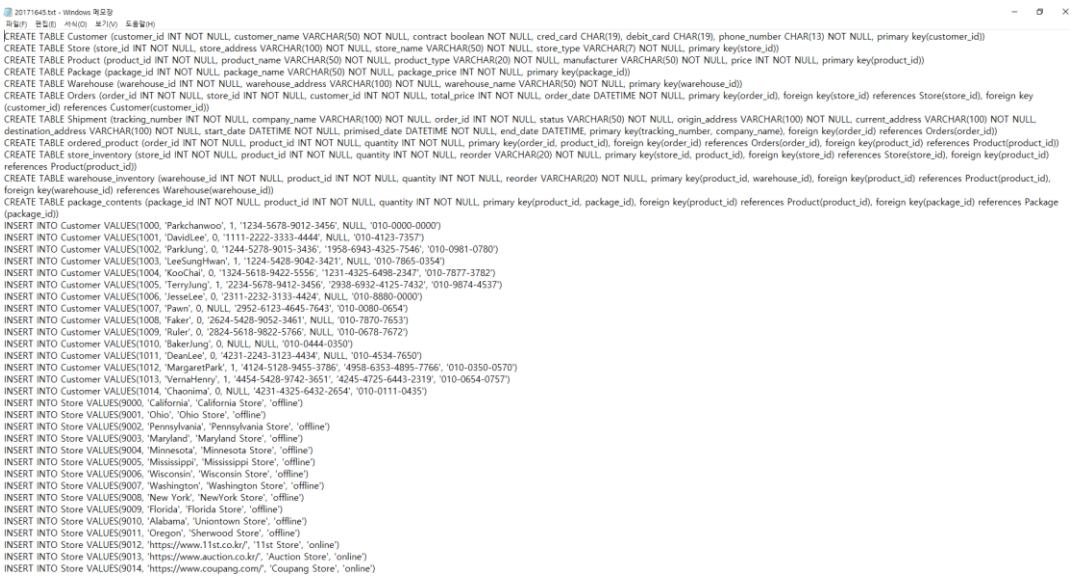
Package와 package_contents, Product 에서 한 제품이 여러 패키지에 들어있을수도, 아무 패키지에 없을수도 있으므로 one-to-zero-one-or-more relationship, 한 패키지에 여러 제품이 있을순 있지만 아무 제품이 없을 수는 없으므로 one-to-one-or-more relationship이 성립한다. 두 relation 모두 package_contents가 상대방으로부터 빌린 foreign key를 primary key로 가지므로, identifying relation이다.

3. Table Creation

위에서 다룬 physical schema를 참조하여 Query에 사용할 데이터베이스를 위한 Table과

각 Table의 Value를 추가한다. 각 Table의 Value는 최소 15개 이상을 집어넣는다.

CRUD query를 위해 text file을 사용한 file input을 사용해야 하므로, txt file로 CRUD 명령어를 작성한다.



```
20171645.txt - Windows 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)
CREATE TABLE Customer (customer_id INT NOT NULL, customer_name VARCHAR(50) NOT NULL, contract boolean NOT NULL, cred_card CHAR(19), debit_card CHAR(19), phone_number CHAR(13) NOT NULL, primary key(customer_id))
CREATE TABLE Store (store_id INT NOT NULL, store_address VARCHAR(100) NOT NULL, store_name VARCHAR(50) NOT NULL, store_type VARCHAR(7) NOT NULL, primary key(store_id))
CREATE TABLE Product (product_id INT NOT NULL, product_name VARCHAR(50) NOT NULL, product_type VARCHAR(20) NOT NULL, manufacturer VARCHAR(50) NOT NULL, price INT NOT NULL, primary key(product_id))
CREATE TABLE Package (package_id INT NOT NULL, package_name VARCHAR(50) NOT NULL, package_price INT NOT NULL, primary key(package_id))
CREATE TABLE Warehouse (warehouse_id INT NOT NULL, warehouse_address VARCHAR(100) NOT NULL, warehouse_name VARCHAR(50) NOT NULL, primary key(warehouse_id))
CREATE TABLE Orders (order_id INT NOT NULL, store_id INT NOT NULL, customer_id INT NOT NULL, total_price INT NOT NULL, order_date DATETIME NOT NULL, primary key(order_id), foreign key(store_id) references Store(store_id), foreign key
(customer_id) references Customer(customer_id))
CREATE TABLE Shipment (tracking_number INT NOT NULL, company_name VARCHAR(100) NOT NULL, order_id INT NOT NULL, status VARCHAR(50) NOT NULL, origin_address VARCHAR(100) NOT NULL, current_address VARCHAR(100) NOT NULL,
destination_address VARCHAR(100) NOT NULL, start_date DATETIME NOT NULL, primised_date DATETIME NOT NULL, end_date DATETIME, primary key(tracking_number, company_name), foreign key(order_id) references Orders(order_id))
CREATE TABLE ordered_product (order_id INT NOT NULL, product_id INT NOT NULL, quantity INT NOT NULL, primary key(order_id, product_id), foreign key(order_id) references Orders(order_id), foreign key(product_id) references Product(product_id))
CREATE TABLE store_inventory (store_id INT NOT NULL, product_id INT NOT NULL, quantity INT NOT NULL, reorder VARCHAR(20) NOT NULL, primary key(store_id, product_id), foreign key(store_id) references Store(store_id), foreign key(product_id)
references Product(product_id))
CREATE TABLE warehouse_inventory (warehouse_id INT NOT NULL, product_id INT NOT NULL, quantity INT NOT NULL, reorder VARCHAR(20) NOT NULL, primary key(product_id, warehouse_id), foreign key(product_id) references Product(product_id),
foreign key(warehouse_id) references Warehouse(warehouse_id))
CREATE TABLE package_contents (package_id INT NOT NULL, product_id INT NOT NULL, quantity INT NOT NULL, primary key(product_id, package_id), foreign key(product_id) references Product(product_id), foreign key(package_id) references Package
(package_id))
INSERT INTO Customer VALUES(1000, 'Parkchanwoo', 1, '1234-5678-9012-3456', NULL, '010-0000-0000')
INSERT INTO Customer VALUES(1001, 'DavidLee', 0, '1111-2222-3333-4444', NULL, '010-4123-7357')
INSERT INTO Customer VALUES(1002, 'ParkJung', 0, '1244-5278-9015-3436', '1958-6943-4325-7546', '010-0981-0780')
INSERT INTO Customer VALUES(1003, 'LeeSungHwan', 1, '1224-5428-9042-3421', NULL, '010-7865-0354')
INSERT INTO Customer VALUES(1004, 'KooChal', 0, '1324-5618-9422-5556', '1231-4325-6498-2347', '010-7877-3782')
INSERT INTO Customer VALUES(1005, 'TerryJung', 1, '2234-5678-9412-3456', '2938-6932-4125-7432', '010-9874-4537')
INSERT INTO Customer VALUES(1006, 'JesseLee', 0, '2311-2232-3133-4424', NULL, '010-8880-0000')
INSERT INTO Customer VALUES(1007, 'Pawn', 0, NULL, '2952-6123-4645-7643', '010-0080-0654')
INSERT INTO Customer VALUES(1008, 'Faker', 0, '2624-5428-9052-3461', NULL, '010-7870-7653')
INSERT INTO Customer VALUES(1009, 'Ruler', 0, '2824-5618-9822-5766', NULL, '010-0678-7672')
INSERT INTO Customer VALUES(1010, 'BakerJung', 0, NULL, NULL, '010-0444-0350')
INSERT INTO Customer VALUES(1011, 'DeanLee', 0, '4231-2243-3123-4434', NULL, '010-4534-7650')
INSERT INTO Customer VALUES(1012, 'MargaretPark', 1, '4124-5128-9455-3786', '4958-6353-4895-7766', '010-0350-0570')
INSERT INTO Customer VALUES(1013, 'VernaHenry', 1, '4454-5428-9742-3651', '4245-4725-6443-2319', '010-0654-0757')
INSERT INTO Customer VALUES(1014, 'Chaonima', 0, NULL, '4231-4325-6432-2654', '010-0111-0435')
INSERT INTO Store VALUES(9000, 'California', 'California Store', 'offline')
INSERT INTO Store VALUES(9001, 'Ohio', 'Ohio Store', 'offline')
INSERT INTO Store VALUES(9002, 'Pennsylvania', 'Pennsylvania Store', 'offline')
INSERT INTO Store VALUES(9003, 'Maryland', 'Maryland Store', 'offline')
INSERT INTO Store VALUES(9004, 'Minnesota', 'Minnesota Store', 'offline')
INSERT INTO Store VALUES(9005, 'Mississippi', 'Mississippi Store', 'offline')
INSERT INTO Store VALUES(9006, 'Wisconsin', 'Wisconsin Store', 'offline')
INSERT INTO Store VALUES(9007, 'Washington', 'Washington Store', 'offline')
INSERT INTO Store VALUES(9008, 'New York', 'NewYork Store', 'offline')
INSERT INTO Store VALUES(9009, 'Florida', 'Florida Store', 'offline')
INSERT INTO Store VALUES(9010, 'Alabama', 'Uniontown Store', 'offline')
INSERT INTO Store VALUES(9011, 'Oregon', 'Sherwood Store', 'offline')
INSERT INTO Store VALUES(9012, 'https://www.11st.co.kr/', '11st Store', 'online')
INSERT INTO Store VALUES(9013, 'https://www.auction.co.kr/', 'Auction Store', 'online')
INSERT INTO Store VALUES(9014, 'https://www.coupang.com/', 'Coupang Store', 'online')
```

생성하는 Table은 위에서 설명한 physical schema의 조건을 그대로 따랐으며, Value는 해당 attribute에 알맞은 값을 임의로 설정해 집어넣는다. 자세한 내용은 20171645.txt 파일을 참고한다.

4. ODBC C language

global

```
 9      const char* host = "localhost";
10      const char* user = "root";
11      const char* pw = "mysql";
12      const char* db = "project";
```

mysql DB와 연결할 정보를 저장하는 문자열로, 각각 host 이름, user 이름, 비밀번호, 연결할 db 이름을 나타낸다.

```
14      void type_one(MYSQL* connection, MYSQL conn);
15      void type_two(MYSQL* connection, MYSQL conn);
16      void type_three(MYSQL* connection, MYSQL conn);
17      void type_four(MYSQL* connection, MYSQL conn);
18      void type_five(MYSQL* connection, MYSQL conn);
19      void type_six(MYSQL* connection, MYSQL conn);
20      void type_seven(MYSQL* connection, MYSQL conn);
```

각 query TYPE에 따른 알맞은 query를 실행하는 함수이다.

main()

```
int main(void) {

    // connect to SQL DBMS

    MYSQL* connection = NULL;
    MYSQL conn;
    MYSQL_RES* sql_result;
    MYSQL_ROW sql_row;

    if (mysql_init(&conn) == NULL)
        printf("mysql_init() error!");

    connection = mysql_real_connect(&conn, host, user, pw, db, 3306, (const char*)NULL, 0);
    if (connection == NULL)
    {
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        return 1;
    }

    else
    {
        printf("Connection Succeed\n");

        if (mysql_select_db(&conn, db))
        {
            printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
            return 1;
        }
```

먼저 mysql_init() 을 통해 MYSQL 오브젝트를 할당하고 초기화한다.

그뒤, mysql_real_connect() 함수를 통해 위에서 설정한 const char* 변수를 활용해 DB와 실제로 연결한다. mysql_init 과 mysql_real_connect시에 적절히 예외처리를 해준다.

```
// FILE IO
FILE* fp;
char buffer[1000];
fp = fopen("20171645.txt", "r");
if (fp == NULL) printf("file open error! \n");
// CREATE table and INSERT tuples
while (fgets(buffer,sizeof(buffer),fp) != NULL)
{
    if (strcmp(buffer, "DROP TABLE package_contents\n") == 0)
    {
        break;
    }
    if (mysql_query(connection, buffer))
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));

    memset(buffer, 0, sizeof(buffer));
}
```

CRUD query 명령어를 저장한 txt 파일을 불러온다. 프로젝트 내의 "20171645.txt" 파일을 불러와, fgets() 함수를 통해 한줄씩 읽어들여 buffer에 저장하고, 해당 query 명령어를 mysql_query() 함수를 사용하여 실행한다. 한번의 query가 끝나면 memset() 함수를 통해

buffer를 다시 비워준다. 만약 읽어들인 query 명령어가 DROP TABLE ~ 명령어라면, 해당 query를 실행하기 전 break 문을 통해 while문을 벗어난다.

```
69          // main query function
70
71          char input;
72          while (1)
73          {
74              printf("₩n------- SELECT QUERY TYPES -------₩n₩n");
75              printf("₩t1. TYPE 1₩n");
76              printf("₩t2. TYPE 2₩n");
77              printf("₩t3. TYPE 3₩n");
78              printf("₩t4. TYPE 4₩n");
79              printf("₩t5. TYPE 5₩n");
80              printf("₩t6. TYPE 6₩n");
81              printf("₩t7. TYPE 7₩n");
82              printf("₩t0. QUIT₩n");
83              printf("₩n--------------------------------₩n");
84              input = _getch();
85              // break function
86              if (input == '0')
87                  break;
88              else if (input == '1')
89                  type_one(connection, conn);
90              else if (input == '2')
91                  type_two(connection, conn);
92              else if (input == '3')
93                  type_three(connection, conn);
94              else if (input == '4')
95                  type_four(connection, conn);
96              else if (input == '5')
97                  type_five(connection, conn);
98              else if (input == '6')
99                  type_six(connection, conn);
100             else if (input == '7')
101                 type_seven(connection, conn);
102         }
```

그런 뒤, 사용자 input을 받을 수 있도록 char 변수 input을 선언한 뒤, 형식에 맞게 user interface를 출력한 뒤 _getch() 함수를 통해 입력을 받는다. 입력받은 값에 따라 알맞은 TYPE의 query를 실행할 수 있도록 if else문을 통해 해당 TYPE의 함수를 호출한다. 만약 입력이 0이라면 while문을 종료한다.

```
104         // drop table and close connection
105         do
106         {
107             if (mysql_query(connection, buffer))
108                 printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
109 
110             memset(buffer, 0, sizeof(buffer));
111         } while (fgets(buffer, sizeof(buffer), fp) != NULL);
112 
113         fclose(fp);
114         mysql_close(connection);
115     }
116 
117     return 0;
118 }
```

만약 user input으로 0을 입력받아 while문을 벗어나면, 먼저 이전의 파일 입출력에서 마지막으로 buffer에 저장되어있던 "DROP TABLE ~" query를 실행한 뒤, fgets() 함수 결과가 null이 될 때까지 한줄씩 파일을 읽고 해당 query를 mysql_query() 함수로 실행한다.

모든 CRUD query가 끝났다면, flose(fp) 및 mysql_close(connection)으로 연 파일과 db와 연결된 connection을 종료해준다.

type_one()

```
120 void type_one(MYSQL* connection, MYSQL conn)
121 {
122     MYSQL_RES* sql_result;
123     MYSQL_ROW sql_row;
124     char track_num[100];
125     char buffer[1000] = "select * from customer c, (select customer_id from orders odr, (select order_id from shipment where company_name = 'USPS' and tracking_number = ";
126     char temp[1000] = " and status = 'Stopped by accident') tracker where odr.order_id = tracker.order_id) cid where c.customer_id = cid.customer_id";
127     int flag = 0;
128     char input;
129 
130     printf("------- TYPE  1  -------\n\n");
131     printf("** Assume the package shipped by USPS with tracking number X is reported to have been destroyed in an accident. Find the contact information for the customer. **\n");
132     printf(" Which X? : ");
133     scanf("%s", track_num);
134     fflush(stdin);
135     strcat(buffer, track_num);
136     strcat(buffer, temp);
137 
138     if (mysql_query(connection, buffer))
139         printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
140     else
141     {
142         sql_result = mysql_store_result(connection);
143         while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
144         {
145             flag = 1;
146             printf("\ncustomer_id : %s\ncustomer_name : %s\nphone_number : %s\n\n", sql_row[0], sql_row[1], sql_row[5]);
147         }
148         if (flag == 0)
149             printf("\nNo Query Result. Please check your tracking number input.\n");
150 
151         mysql_free_result(sql_result);
152     }
153 
154     if (flag == 0)
155         return;
```

형식에 맞게 user interface를 출력한 뒤, scanf() 를 통해 찾고자 하는 tracking number를 user input으로 받고, 이를 통해 다음 query 명령어를 실행한다.

```
select * from customer c, (select customer_id from orders odr, (select order_id from shipment where company_name = 'USPS' and tracking_number = X and status = 'Stopped by accident') tracker where odr.order_id = tracker.order_id) cid where c.customer_id = cid.customer_id;
```

USPS에 의해 운송되던 tracking number가 X이며 배송의 배송상태가 'Stopped by accident' 인 배송의 고객의 customer_id를 2중 query 중 내부의 query로부터 가져오고, 외부 query는 해당 customer_id를 가진 customer의 정보를 select 해오는 query이다. 이를 통

해 사고로 파손된 tracking number X를 갖는 USPS의 배송을 시킨 고객의 정보를 얻을 수 있다.

mysql_store_result() 함수를 통해 query 결과를 가져오고, mysql_fetch_row() 함수를 통해 해당 결과를 각 row 별로 sql_row에 따로 저장한다. 이때 flag 변수를 통해 query 결과가 하나 이상 있다면 해당 query 결과를 전부 형식에 맞게 출력하고, 하나도 없다면

`No Query Result. Please check your tracking number input` 를 출력해준다.

해당 query를 수행한 결과는

customer_id	customer_name	contract	cred_card	debit_card	phone_number	customer_id
1001	DavidLee	0	1111-2222-3333-4444	NULL	010-4123-7357	1001

와 같은 형태로 반환되는데, TYPE 1 query가 요구하는 정보는 customer의 contact information 이므로 고객의 id, 이름, 전화번호만을 출력한다.

현재 구성한 table에서 회사가 USPS이면서 배송상태가 'Stopped by accident' 인 배송의 경우는 tracking number가 7014 하나뿐인데, 이를 입력한 결과는 아래와 같다.

```
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        6. TYPE 6
        7. TYPE 7
        0. QUIT

-----------------------------------
-------  TYPE  1  -------

** Assume the package shipped by USPS with tracking number X is reported to have been destroyed in an accident. Find the
 contact information for the customer. **
 Which X? : 7014

customer_id : 1001
customer_name : DavidLee
phone_number : 010-4123-7357
```

```
    // subtype

    printf("---------- Subtypes in TYPE 1 ----------\n");
    printf("\t1. TYPE 1-1\n\n");
    input = _getch();

    if (input == '0')
        return;
    else if (input == '1')
    {
        printf("------- TYPE 1-1 -------\n\n");
        printf("** Find the contents of that shipment and create a new shipment of replacement items. **\n");

        strcpy(buffer, "select k.product_name, m.quantity from product k, (select w.product_id, w.quantity from ordered_product w, (select * from shipment where tracking_number = ");
        strcat(buffer, track_num);
        strcat(buffer, ") z where w.order_id = z.order_id) m where k.product_id = m.product_id");

        if (mysql_query(connection, buffer))
            printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        else
        {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("product_name : %s\t\tproduct_quantity : %s\n", sql_row[0], sql_row[1]);
            }

            mysql_free_result(sql_result);
        }
```

이후 TYPE 1의 subtype인 TYPE 1-1 query를 할 지 입력받는다.

동일하게 _getch() 함수를 사용하여 사용자에게 입력받고, 만약 0을 입력받는다면 SELECT

QUERY TYPES 화면으로 돌아가고 1을 입력받으면 SUBTYPE 1-1 query를 수행하도록 한다. TYPE 1-1 query의 경우 아래 query 명령어를 실행한다.

```
select k.product_name, m.quantity from product k, (select w.product_id, w.quantity from ordered_product w, (select * from shipment where tracking_number = X) z where w.order_id = z.order_id) m where k.product_id = m.product_id
```

위의 X는 TYPE 1에서 입력받은 X와 동일한 값으로, query 내용은 3중 query중 가장 내부 query를 통해 tracking_number가 X인 shipment 정보를 가져오고, 그 밖 내부 query를 통해 해당 shipment와 order_id가 동일한 ordered_product의 product_id, quantity를 가져와서, 가장 밖 query를 통해 product_id가 동일한 product의 이름과, 이전에 가져온 quantity를 출력한다.

그 결과, tracking number가 X인 shipment의 배송 내용물의 이름과 그 개수를 출력한다.

그런다음, 해당 정보를 바탕으로 새로운 shipment를 replacement items로 준비하기 위해, 아래 query를 수행한다.

```
select * from shipment order by tracking_number desc
```

해당 query를 실행한 결과, 모든 shipment 내용을 tracking_number 순으로 내림차순 정렬해 가져오는데,

sql_result = mysql_store_result(connection) 와

sql_row = mysql_fetch_row(sql_result) 를 통해 해당 query의 첫 줄만 가져오고,

strcpy(temp, sql_row[0]) 를 통해 가장 큰 tracking number를 구해온다.

현재 구성된 table에서 tracking_number는 7000부터 시작해서 순차적으로 1씩 올라 7001, 7002, ... 의 형태로 저장되는 형태이므로, 가장 큰 tracking number를 구해 1 더한 값을 할당하면 겹치지 않는 tracking number를 구할 수 있다. atoi() 및 sprintf() 함수를 통해 가장 큰 tracking number + 1의 값을 문자열 형태로 temp에 저장하고, 아래 query를 수행한다.

```
INSERT INTO Shipment(tracking_number, company_name, order_id, status, origin_address, current_address, destination_address, start_date, primised_date, end_date) SELECT K,
```

```
s.company_name, s.order_id, 'Information', s.origin_address, s.origin_address, s.destination_address, '2022:05:30 12:00:00', '2022:06:02 12:00:00', NULL from shipments where s.tracking_number = X
```

.동일하게 X는 이전의 tracking number이고, K는 바로 전에 구한 biggest tracking number + 1 값이다.

이를 query를 통해, 이전의 tracking number X를 갖는 shipment에서 company_name, order_id, origin_address, destination_address를 가져오고, K를 새로운 tracking number로, 현재상태를 "information", 즉 배송정보 접수로, 현재 위치를 origin address와 동일하게, 배송시작 시간, 배송예정 시간을 임의로 설정한 값으로 설정하여, INSERT INTO query 명령어를 통해 Shipment에 새로운 tuple 값을 생성해 집어넣는다.

시간 설정의 경우 time.h library를 활용하여 현재 시간으로 설정해야 할지 고민했으나 프로젝트에 중요한 내용이 아니라고 생각해 임의의 값으로 설정했다.

해당 query 명령어를 통해 tuple을 집어넣는데 성공하면 성공했다고 출력한다.

subquery 실행 결과는 아래와 같이 나온다.

```
---------- Subtypes in TYPE 1 ----------
        1. TYPE 1-1

------- TYPE 1-1 -------

** Find the contents of that shipment and create a new shipment of replacement items. **
product_name : K590 LED          product_quantity : 2
product_name : G340 LIGHTSPEED          product_quantity : 2
Succesfully added new shipment
```

tracking_number	company_name	order_id	status	origin_address	current_address	destination_address	start_date	primised_date	end_date
7015	USPS	4026	Information	Address, 2014	Address, 2014	Address, 2019	2022-05-30 12:00:00	2022-06-02 12:00:00	NULL
7014	USPS	4026	Stopped by accident	Address, 2014	Address, 9999	Address, 2019	2022-05-25 22:00:00	2022-06-03 12:00:00	NULL

새로 생성한 tuple이 정상적으로 들어간 모습을 확인할 수 있다.

type_two

```
void type_two(MYSQL* connection, MYSQL conn)
{
    MYSQL_RES* sql_result;
    MYSQL_ROW sql_row;
    char buffer[1000] = "select * from customer q, (select temp.customer_id from (select customer_id, sum(total_price) as s from orders where DATE(order_date) between '2021-01-01' and '2021-12-31' group by cu
    char result[10];
    char input;

    printf("------- TYPE 2 -------\n\n");
    printf("** Find the customer who has bought the most (by price) in the past year. **\n");

    if (mysql_query(connection, buffer))
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
    else
    {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf("customer_id : %s\tcustomer_name : %s\tcontract : %s\tcred_card : %s\tdebit_card : %s\tphone_number : %s\t\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4], sql_row[5]);
            strcpy(result, sql_row[0]);
        }
        mysql_free_result(sql_result);
    }
```

아래 query를 실행한다.

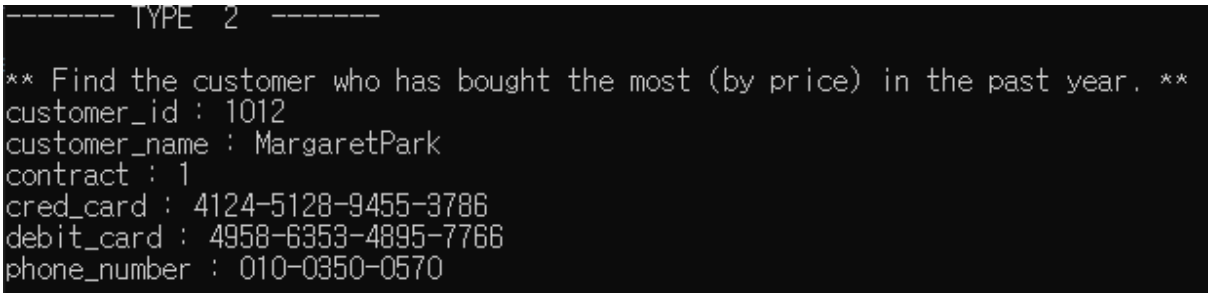
```
select * from customer q, (select temp.customer_id from (select customer_id,
```

```
sum(total_price) as s from orders where DATE(order_date) between '2021-01-01' and '2021-12-31' group by customer_id) temp order by temp.s desc limit 1) res where q.customer_id = res.customer_id
```

위 query는 orders의 order_date의 값이 작년, 즉 2021-01-01 부터 2021-12-31인 order 들을 customer_id를 기준으로 group화 하여 각 customer 별로 총액을 구한 뒤 내림차순 정렬한뒤 가장 위의 1개 값을 구하여, 그 customer의 정보를 출력한다.

그 결과, 작년에 주문한 고객들 중 가격 기준으로 가장 많이 구매한 고객 1명을 구한다.

이때, 해당 고객의 customer_id를 따로 저장한다. (이후 subtype에서 사용)




```
// subtype

printf("---------- Subtypes in TYPE 2 ----------\n");
printf("\t1. TYPE 2-1\n\n");
input = _getch();

if (input == '0')
    return;
else if (input == '1')
{
    printf("------- TYPE 2-1 -------\n\n");
    printf("** Find the product that the customer bought the most. **\n");

    strcpy(buffer, "select * from product pd, (select product_id from ordered_product op, (select order_id, customer_id from orders where orders.customer_id = ");
    strcat(buffer, result);
    strcat(buffer, ") od where op.order_id = od.order_id order by op.quantity desc limit 1) pdid where pd.product_id = pdid.product_id");

    if (mysql_query(connection, buffer))
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
    else
    {
        printf("product that the customer above bought the most\n");
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf("product_id : %s\tproduct_name : %s\n", sql_row[0], sql_row[1]);
        }

        mysql_free_result(sql_result);
    }
}
```

이후, subtype 2-1을 TYPE 1-1과 동일한 과정으로 입력받고, 아래 query를 수행한다.

```
select * from product pd, (select product_id from ordered_product op, (select order_id, customer_id from orders where orders.customer_id = X ) od where op.order_id = od.order_id order by op.quantity desc limit 1) pdid where pd.product_id = pdid.product_id
```

이 query는 이전 query에서 구한 customer_id를 바탕으로, 해당 고객이 구매한 물건을 orderd_product의 quantity를 기준으로 내림차순 정렬한 뒤 최상위 1개의 값을 구하여 그 고객이 가장 많이 구매한 물건의 product_id를 구한 뒤, 그 product_id의 정보를 출력한다.

subquery 실행 결과는 아래와 같다.

```
---------- Subtypes in TYPE 2 ----------
        1. TYPE 2-1

------- TYPE 2-1 -------

** Find the product that the customer bought the most. **
product that the customer above bought the most
product_id : 2002
product_name : Iphone SE3
product_type : smartphone
manufacturer: Apple
price : 990
```

type_three()

```
272  void type_three(MYSQL* connection, MYSQL conn)
273  {
274      MYSQL_RES* sql_result;
275      MYSQL_ROW sql_row;
276      char buffer[1000] = "select p.*, sum(op.quantity), p.price * sum(op.quantity) as dollar from product p, ordered_product op, orders o where DATE(o.order_date) between '2021-01-01' and '2021-12-31' and o.or
277      char input;
278      char result[10];
279      int count = 0;
280
281      printf("------- TYPE  3  -------\n\n");
282      printf("** Find all products sold in the past year. **\n");
283
284      if (mysql_query(connection, buffer))
285          printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
286      else
287      {
288          sql_result = mysql_store_result(connection);
289          while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
290          {
291              printf("product_id : %s\tproduct_name : %s\n", sql_row[0], sql_row[1]);
292          }
293          mysql_free_result(sql_result);
294      }
```

아래 query를 수행한다.

```
select p.*, sum(op.quantity), p.price * sum(op.quantity) as dollar from product p, ordered_product op, orders o where DATE(o.order_date) between '2021-01-01' and '2021-12-31' and o.order_id = op.order_id and op.product_id = p.product_id group by product_id order by dollar desc
```

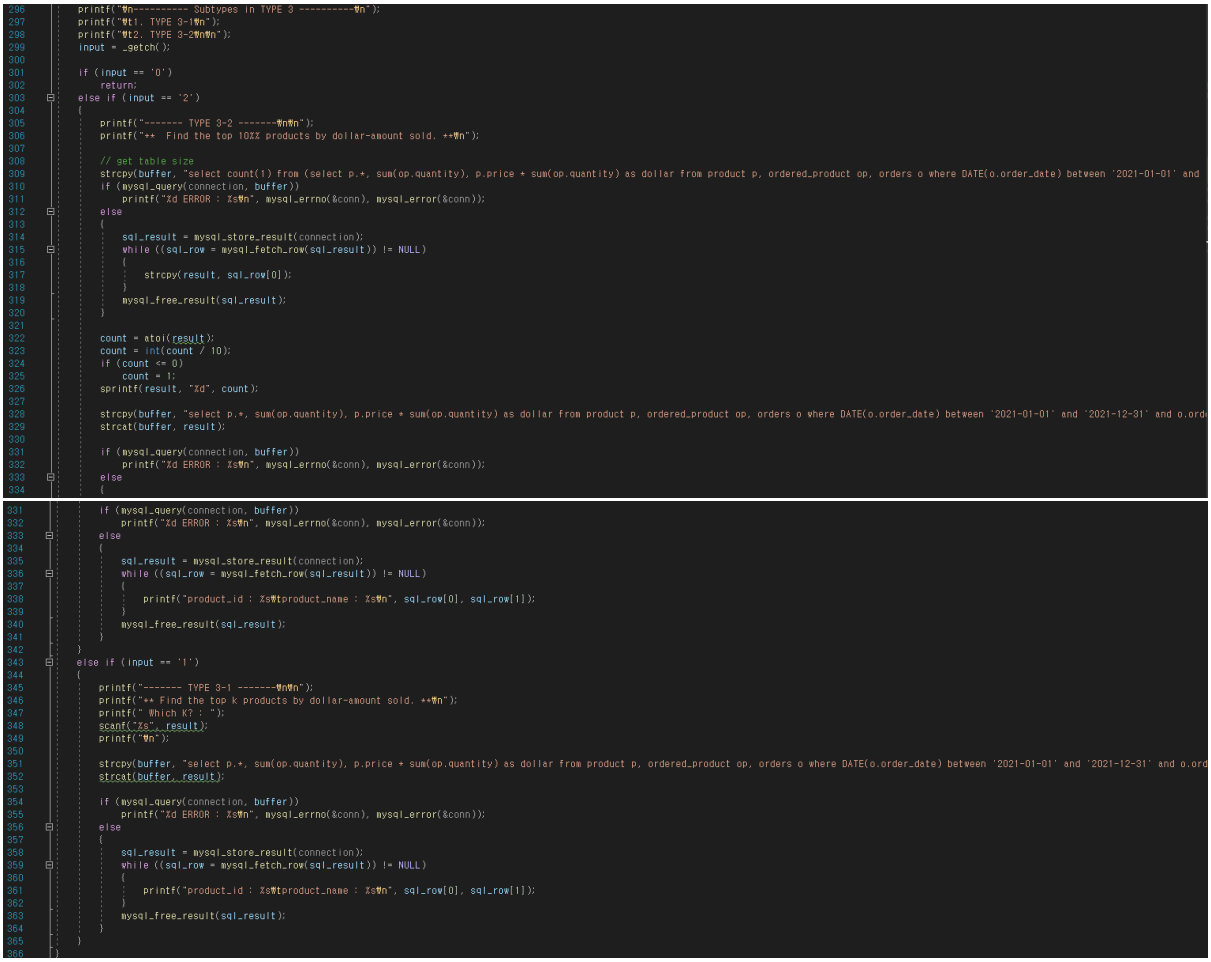
이 query는 order_date가 작년인 order의 order_id와 동일한 order_id를 갖는 ordered_product의 product_id를 구한뒤 ordered_product의 quantity * price의 sum을 product_id 를 기준으로 group화 하여 각 물품이 총액 얼마나 팔렸는지에 대한 정보를 구한 뒤 내림차순 정렬하고, 동일한 product_id를 가진 product의 정보를 출력한다.

그 결과, 작년에 판매된 모든 상품들을 판매 총액 기준 내림차순정렬해 전부 가져와, 출력해준다. 그 결과는 아래와 같다.

```
------- TYPE  3  -------

** Find all products sold in the past year. **
product_id : 2000       product_name :          Galaxy S22      total sales($) : 2700
product_id : 2002       product_name :          Iphone SE3      total sales($) : 1980
product_id : 2001       product_name :       GalaxyNote 10+     total sales($) : 800
product_id : 2012       product_name :         Dyson VT20T9     total sales($) : 690
product_id : 2007       product_name : Basilisk X HyperSpeed    total sales($) : 480
product_id : 2009       product_name :               Z-2100     total sales($) : 330
product_id : 2005       product_name :            K590 LED      total sales($) : 300
product_id : 2014       product_name :      FC900R PD Brown     total sales($) : 280
```

이때 query 결과는 product의 모든 정보를 가져오지만, 모든 내용을 출력하면 출력화면이 너무 지저분해져 product_id, product_name, 판매 총액 3가지만 출력하도록 했다.



```
    printf("\n---------- Subtypes in TYPE 3 ----------\n");
    printf("\t1. TYPE 3-1\n");
    printf("\t2. TYPE 3-2\n\n");
    input = _getch();

    if (input == '0')
        return;
    else if (input == '2')
    {
        printf("------- TYPE 3-2 -------\n\n");
        printf("++  Find the top 10XX products by dollar-amount sold. ++\n");

        // get table size
        strcpy(buffer, "select count(1) from (select p.*, sum(op.quantity), p.price * sum(op.quantity) as dollar from product p, ordered_product op, orders o where DATE(o.order_date) between '2021-01-01' and '
        if (mysql_query(connection, buffer))
            printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        else
        {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                strcpy(result, sql_row[0]);
            }
            mysql_free_result(sql_result);
        }

        count = atoi(result);
        count = int(count / 10);
        if (count <= 0)
            count = 1;
        sprintf(result, "%d", count);

        strcpy(buffer, "select p.*, sum(op.quantity), p.price * sum(op.quantity) as dollar from product p, ordered_product op, orders o where DATE(o.order_date) between '2021-01-01' and '2021-12-31' and o.ord
        strcat(buffer, result);

        if (mysql_query(connection, buffer))
            printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        else
        {
        if (mysql_query(connection, buffer))
            printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        else
        {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("product_id : %s\tproduct_name : %s\n", sql_row[0], sql_row[1]);
            }
            mysql_free_result(sql_result);
        }
    }
    else if (input == '1')
    {
        printf("------- TYPE 3-1 -------\n\n");
        printf("++ Find the top k products by dollar-amount sold. ++\n");
        printf(" Which K? : ");
        scanf("%s", result);
        printf("\n");

        strcpy(buffer, "select p.*, sum(op.quantity), p.price * sum(op.quantity) as dollar from product p, ordered_product op, orders o where DATE(o.order_date) between '2021-01-01' and '2021-12-31' and o.ord
        strcat(buffer, result);

        if (mysql_query(connection, buffer))
            printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        else
        {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("product_id : %s\tproduct_name : %s\n", sql_row[0], sql_row[1]);
            }
            mysql_free_result(sql_result);
        }
    }
}
```

이후, subtype query를 수행한다. 1을 입력받은 경우, 먼저 사용자 input K를 받은 뒤

```
select p.*, sum(op.quantity), p.price * sum(op.quantity) as dollar from product p, ordered_product op, orders o where DATE(o.order_date) between '2021-01-01' and '2021-12-31' and o.order_id = op.order_id and op.product_id = p.product_id group by product_id order by dollar desc limit K
```

query를 통해 TYPE 3의 결과에서 limit K를 통해 상위 K개의 결과만 출력해준다.

2를 입력받은 경우, 먼저

```
select count(1) from (select p.*, sum(op.quantity), p.price * sum(op.quantity) as dollar from product p, ordered_product op, orders o where DATE(o.order_date) between '2021-01-01' and '2021-12-31' and o.order_id = op.order_id and op.product_id = p.product_id group by product_id order by dollar desc) tb
```

를 통해, TYPE 3 결과에서 tuple의 총 개수를 구한 뒤, 그 결과를 result에 저장한다.

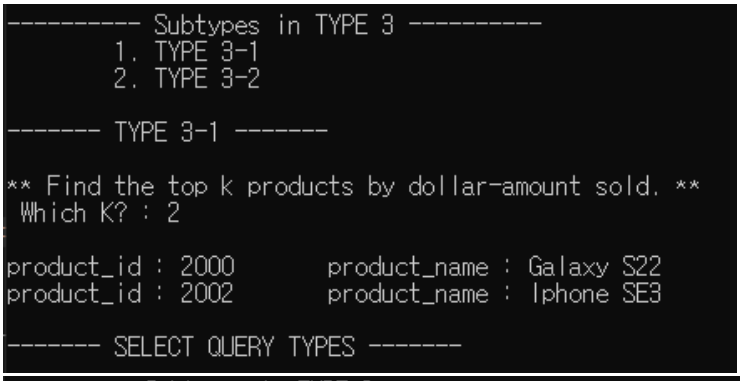
이후 atoi 함수를 통해 결과를 정수로 바꾼 뒤, 상위 10% product의 개수를 얻기 위해 그 값을 10으로 나눈다. C에서 나눈 값을 int로 type casting할경우 버림이 되는데, 이럴

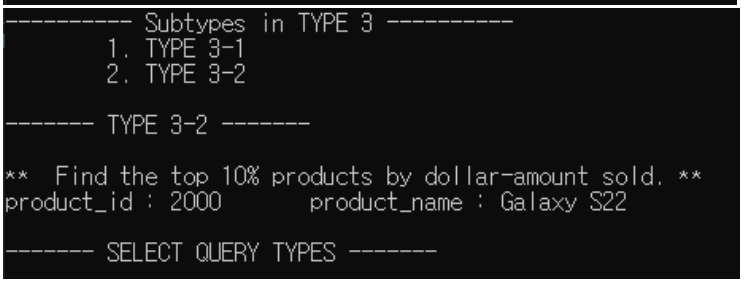
경우 결과가 0인 경우가 생기므로 if문을 통해 결과가 0일경우 최상위 1개를 출력하도록 설정한다. 이후

```
select p.*, sum(op.quantity), p.price * sum(op.quantity) as dollar from product p, ordered_product op, orders o where DATE(o.order_date) between '2021-01-01' and '2021-12-31' and o.order_id = op.order_id and op.product_id = p.product_id group by product_id order by dollar desc limit X
```

를 통해 바로 전에 계산한 상위 10%를 출력한다.

실행 결과는 아래와 같다.





type_four()

```
void type_four(MYSQL* connection, MYSQL conn)
{
    MYSQL_RES* sql_result;
    MYSQL_ROW sql_row;
    char buffer[1000] = "select p.*, sum(op.quantity) from product p, ordered_product op, orders o where DATE(o.order_date) between '2021-01-01' and '2021-12-31' and o.order_id = op.order_id and op.product_id
    char input;
    char result[10];
    int count = 0;

    printf("------- TYPE  4  -------\n\n");
    printf("++ Find all products by unit sales in the past year. ++\n");

    if (mysql_query(connection, buffer))
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
    else
    {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf("product_id : %s\tproduct_name : %20s\ttotal sales($) : %s\n", sql_row[0], sql_row[1], sql_row[6]);
        }
        mysql_free_result(sql_result);
    }

    printf("\n---------- Subtypes in TYPE 4 ----------\n");
    printf("\t1. TYPE 4-1\n");
    printf("\t2. TYPE 4-2\n\n");
    input = _getch();

    if (input == '0')
        return;
    else if (input == '2')
    {
        printf("------- TYPE 4-2 -------\n\n");
        printf("**  Find the top 10%% products by unit sales. **\n");
```

TYPE 4는 query와 2개의 subquery 모두 TYPE3와 거의 모든 면에서 동일하다. 유일한 차이점은, TYPE 4의 query 내용은 아래와 같은데

```
select p.*, sum(op.quantity) from product p, ordered_product op, orders o where DATE(o.order_date) between '2021-01-01' and '2021-12-31' and o.order_id = op.order_id and op.product_id = p.product_id group by product_id order by sum(op.quantity) desc
```

유일한 차이점은 order by sum(op.quantity)로, 기존의 sum(op.quantity) * price 로 물건의 판매 총액을 기준으로 정렬한 것을, 물건의 unit sales, 즉 판매 개수 기준으로 정렬한다는 것이 차이점이다. 그 외에는 전부 동일하다.

실행 결과는 각각 아래와 같다.

```
------- TYPE  4  -------

** Find all products by unit sales in the past year. **
product_id : 2000       product_name :           Galaxy S22     total sales(quantity) : 3
product_id : 2005       product_name :             K590 LED     total sales(quantity) : 3
product_id : 2002       product_name :            Iphone SE3    total sales(quantity) : 2
product_id : 2007       product_name : Basilisk X HyperSpeed    total sales(quantity) : 2
product_id : 2001       product_name :        GalaxyNote 10+    total sales(quantity) : 1
product_id : 2009       product_name :                Z-2100    total sales(quantity) : 1
product_id : 2012       product_name :          Dyson VT20T9    total sales(quantity) : 1
product_id : 2014       product_name :       FC900R PD Brown    total sales(quantity) : 1

---------- Subtypes in TYPE 4 ----------
        1. TYPE 4-1
        2. TYPE 4-2

------- TYPE 4-1 -------

** Find the top k products by unit sales. **
 Which K? : 2

product_id : 2000       product_name :           Galaxy S22     total sales(quantity) : 3
product_id : 2005       product_name :             K590 LED     total sales(quantity) : 3
```

```
------- TYPE 4-2 -------

**  Find the top 10% products by unit sales. **
product_id : 2000       product_name :           Galaxy S22     total sales(quantity) : 3
```

type_five()

```
463  void type_five(MYSQL* connection, MYSQL conn)
464  {
465      MYSQL_RES* sql_result;
466      MYSQL_ROW sql_row;
467      char buffer[1000] = "select p.* from product p, store_inventory si, store s where s.store_address = 'California' and s.store_id = si.store_id and si.quantity = 0 and si.product_id = p.product_id";
468
469      printf("------- TYPE  5  -------\n\n");
470      printf("** Find those products that are out-of-stock at every store in California. **\n");
471
472      if (mysql_query(connection, buffer))
473          printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
474      else
475      {
476          sql_result = mysql_store_result(connection);
477          while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
478          {
479              printf("product_id : %s\tproduct_name : %s\tproduct_type : %s\tmanufacturer: %s\tprice : %s\n\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4]);
480          }
481          mysql_free_result(sql_result);
482      }
483  }
```

아래 query를 수행한다.

```
select p.* from product p, store_inventory si, store s where s.store_address = 'California' and s.store_id = si.store_id and si.quantity = 0 and si.product_id = p.product_id
```

이 query는 store_address가 California 이면서 그 store_id와 동일한 store_id를 갖는 store_inventory의 quantity가 0이며 그 store_inventory의 product_id와 동일한 product_id를 갖는 product의 모든 정보를 select 해온다.

그 실행 결과는 아래와 같다.

```
------- TYPE  5  -------

** Find those products that are out-of-stock at every store in California. **
product_id : 2001
product_name : GalaxyNote 10+
product_type : smartphone
manufacturer: Samsung
price : 800

product_id : 2002
product_name : Iphone SE3
product_type : smartphone
manufacturer: Apple
price : 990
```

type_six()

```
void type_six(MYSQL* connection, MYSQL conn)
{
    MYSQL_RES* sql_result;
    MYSQL_ROW sql_row;
    char buffer[1000] = "select tracking_number, company_name, op.order_id, p.product_name, op.quantity from shipment sh, ordered_product op, product p where sh.primised_date < sh.end_date and sh.order_id =

    printf("------- TYPE  6  -------\n\n");
    printf("** Find those packeges that were not delivered within the promised time. **\n");

    if (mysql_query(connection, buffer))
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
    else
    {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf("tracking_number of shipment : %s\tcompany_name of shipment : %s\norder_id : %s\tordered_product : %s\t\tquantity : %s\n\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4]);
        }
        mysql_free_result(sql_result);
    }
}
```

아래 query를 수행한다.

```
select tracking_number, company_name, op.order_id, p.product_name, op.quantity from shipment sh, ordered_product op, product p where sh.primised_date < sh.end_date and sh.order_id = op.order_id and op.product_id = p.product_id group by order_id
```

이 query는 shipment의 promised_date보다 end_date가 늦은, 즉 약속된 시간보다 배송이 늦게 완료된 shipment의 order_id, 이를통해 해당 order의 내용물과 주문한 상품 수를 얻어 tracking_number, company_name, order_id, product_name, quantity를 select 해오는 query 문이다. 그 결과, 약속된 시간보다 배송이 늦게 된 배송과 그에 대응하는 주문, 그 주문의 내용물인 상품의 이름을 가져온다.

실행 결과는 아래와 같다.

```
------- TYPE  6  -------

** Find those packages that were not delivered within the promised time. **
tracking_number of shipment : 7000      company_name of shipment : QuickDelivery
order_id : 4012 ordered_product : Iphone SE3            quantity : 2

tracking_number of shipment : 7006      company_name of shipment : FastDelivery
order_id : 4018 ordered_product : Companion 2 series 3          quantity : 1

tracking_number of shipment : 7007      company_name of shipment : FastDelivery
order_id : 4019 ordered_product : Z-2100                quantity : 1

tracking_number of shipment : 7008      company_name of shipment : FastDelivery
order_id : 4020 ordered_product : Dell UltraSharp U3219Q                quantity : 1
```

type_seven()

아래 query를 수행한 결과를 출력한다.

select * from customer c, orders o, ordered_product op, product p where DATE(o.order_date) between '2022-05-01' and '2022-06-01' and o.customer_id = c.customer_id and o.order_id = op.order_id and op.product_id = p.product_id order by c.customer_id

위 query는 order_date가 저번달인 2022-05-01부터 2022-06-01 사이인 모든 order의 order 정보와 해당 order를 주문한 customer, 해당 order의 내용물 ordered_product의 각 product의 정보를 전부 select 해오는 query 문이다. 출력시 화면의 가독성을 위해, 필요한 내용인 고객의 id, 고객의 이름, 주문 id, 주문의 총액, 주문의 상품 이름, 주문한 상품 개수를 출력해준다.

해당 query의 결과는 아래와 같다.

```
------- TYPE  7  -------

** Generate the bill for each customer for the past month. **
customer_id : 1009      customer_name : Ruler
order_id : 4023         total_price : 300
order_content :          K70 Lux Red    order_quantity : 1

customer_id : 1011      customer_name : DeanLee
order_id : 4011         total_price : 800
order_content :       GalaxyNote 10+    order_quantity : 1

customer_id : 1013      customer_name : VernaHenry
order_id : 4013         total_price : 1200
order_content :          Odyssey xm3    order_quantity : 3

customer_id : 1014      customer_name : Chaonima
order_id : 4014         total_price : 500
order_content :         BenQ EW3290U    order_quantity : 1
```