

CSE4110 – Database System

Project2. Normalization and Query Processing

Package Delivery System

20191559 강상원

1. Project 개요

Project 1에서 개발한 택배 회사의 데이터베이스 설계의 Logical Schema를 BCNF 정규화하는 과정을 거친다. 이후 Physical Schema diagram에 도메인, 제약 조건, null 허용 여부 등의 정보를 기입한다. 명세서에 주어진 7가지 쿼리를 작성하여 ODBC로 MySQL과 Visual Studio를 연결하여 이를 제어할 수 있는 사용자 프로그램을 개발한다.

2. BCNF Decomposition

Project 1에서 설계한 택배 회사의 Relationship Schema는 다음과 같았다.

1. Customer (customer_id[PK], name, address, account_number)
2. Package (package_id[PK], customer_id[FK], recipient_id[FK], type, weight, service, ship_date, delivery_date, contents)

배송품 entity에는 Customer 및 Recipient entity와의 관계를 설정하는 두 개의 외래 키인 customer_id 및 recipient_id가 있다. 이러한 외래 키는 일대다 카디널리티를 반영하여 각 배송품이 단일 고객 및 수신자와 연결되도록 한다.

3. Recipient (recipient_id[PK], name, address)
4. Shipment (shipment_id[PK], shipment_enum[PK], location_id[FK], timestamp, status)
Shipment entity에는 두 개의 외래 키인 package_id 및 shipment_id가 있어 이를 Package 및 Shipment entity에 연결한다. 이러한 외래 키는 각 배송 이벤트가 일대다 카디널리티를 사용하여 다중 배송품 및 위치와 연결되도록 한다.

5. Contain (shipment_id[PK, FK], shipment_enum[PK, FK], package_id[PK, FK])
Contain entity에는 Shipment의 외래 키인 shipment_id, shipment_enum과 Package의 외래 키인 package_id가 있다. 이러한 외래 키는 각 배송 이벤트에 여러 package들이 들어가고, 각 package들이 여러 배송 단계에 속함을 나타낸다.

6. Location (location_id[PK], location_name, location_type, location_contact, capacity)
7. Billing (billing_id[PK], customer_id[FK], package_ID[FK], amount, billing_date,

billing_type)

Billing entity에는 고객 및 배송품 entity에 연결되는 두 개의 외래 키인 customer_id 및 package_id가 있다. customer_id 외래 키는 Customer와 일대다 관계를 설정하고 package_id 외래 키는 Package와 일대일 관계를 설정한다.

BCNF(Boyce-Codd) Decomposition을 위 Logical Schema에 적용 시도한다.

Relation F에 대해 모든 functional dependency가 ①F의 Superkey에만 종속되거나 ②Trivial 하지 않다면 Decomposition의 필요가 있다.

BCNF dependency를 체크하면, 다음과 같다.

- Customer (customer_id[PK], name, address, account_number)의 FD:

$(\text{customer_id}) \rightarrow (\text{name}, \text{address}, \text{account_number})$ 이외의 FD가 존재하지 않는다. 좌항인 customer_id는 Primary key이므로, BCNF를 만족하여 Decomposition의 필요가 없다. name은 동명이인이 있을 수 있기 때문에 종속성을 따로 만들지 않는다.

- Package (package_id[PK], customer_id[FK], recipient_id[FK], type, weight, service, ship_date, delivery_date, contents)

$(\text{package_id}) \rightarrow (\text{customer_id}, \text{recipient_id}, \text{type}, \text{weight}, \text{service}, \text{ship_date}, \text{delivery_date}, \text{contents})$ 외에 다른 FD가 없고, package_id가 Primary key이므로 BCNF를 만족한다.

- Recipient (recipient_id[PK], name, address)

$(\text{recipient_id}) \rightarrow (\text{name}, \text{address})$ 외에 다른 FD가 없고, recipient_id가 Primary key이므로 BCNF를 만족한다. address 값에 따른 FD가 없는 이유는 한 주소(집)에 여러 명의 수신자가 있을 수 있기 때문이다.

- Shipment (shipment_id[PK], shipment_enum[PK], location_id[FK], timestamp, status)

$(\text{shipment_id}, \text{shipment_enum}) \rightarrow (\text{location_id}, \text{timestamp}, \text{status})$ 외에 다른 FD가 없고, (shipment_id, shipment_enum)이 Primary key이므로 BCNF를 만족한다.

- Contain (shipment_id[PK, FK], shipment_enum[PK, FK], package_id[PK, FK])

마찬가지로 Primary key 외에 다른 FD가 없으므로 BCNF를 만족한다.

- Location (location_id[PK], location_name, location_type, location_contact, capacity)

마찬가지로 Primary key 외에 다른 FD가 없으므로 BCNF를 만족한다.

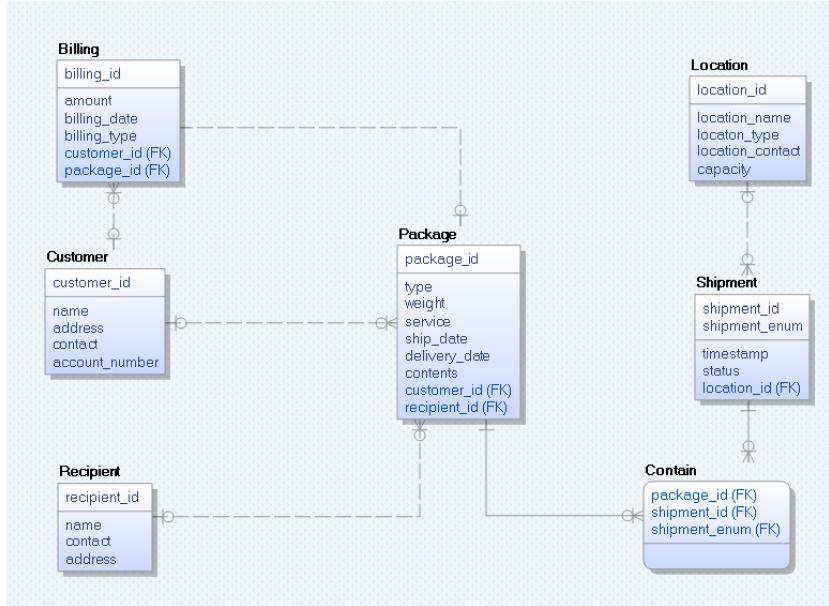
location_contact는 여러 장소/운송수단의 관리자/운전자가 같을 수 있기 때문에 종속성을

만들지 않는다.

7. Billing (billing_id[PK], customer_id[FK], package_ID[FK], amount, billing_date, billing_type)

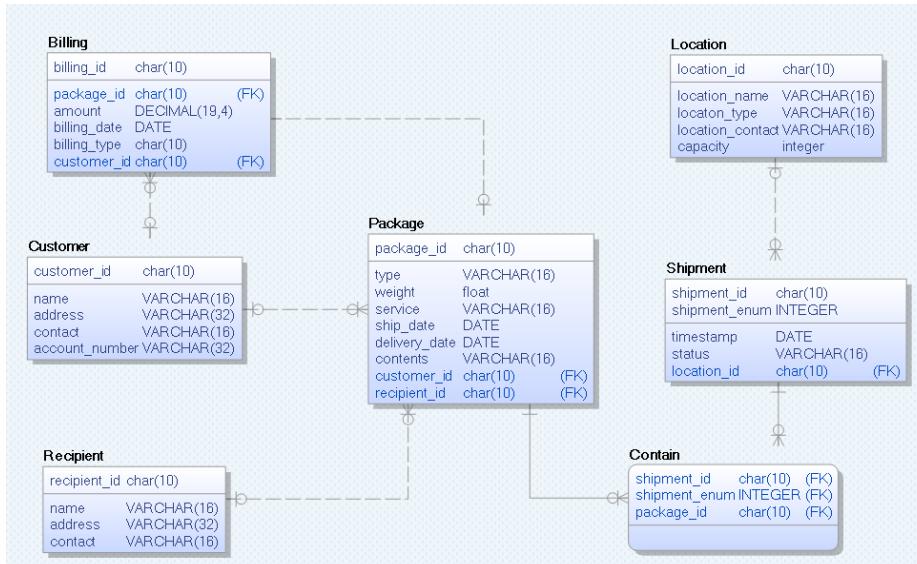
마찬가지로 Primary key 외에 다른 FD가 없으므로 BCNF를 만족한다. 고객의 매 거래마다 거래량, 거래 방식, 가격은 달라지므로 추가 FD는 생기지 않는다.

따라서 기존 Logical Schema는 모두 BCNF dependency를 만족하여, Decomposition 한 결과는 이전과 동일하다.



3. Physical Schema Diagram

위 단계에서 만든 Logical Schema diagram으로, Physical Schema diagram을 만들어야 한다.



모든 entity와 relationship이 BCNF를 만족하였기 때문에 decomposition은 없고, 대신 NULL 허용 여부, data type 등의 정보를 추가하였다. 각 Table에 대한 설명은 아래와 같다.

```
CREATE TABLE Customer (
    customer_id CHAR(10) NOT NULL,
    name VARCHAR(16) NULL,
    address VARCHAR(32) NULL,
    contact VARCHAR(16) NULL,
    account_number VARCHAR(32) NULL,
    PRIMARY KEY (customer_id)
);
```

Customer의 Primary key인 customer_id는 NULL이어서는 안되고, 고정 길이 10의 문자열로 정의한다.

이름, 주소, 계좌번호와 같은 값은 각 field에 맞는 충분한 길이의 가변 문자열로 정의하였다. 보낸 이의 주소를 기입하지 않는 익명 택배거나, 결제가 필요 없는 택배 서비스도 있을 수 있으므로 NULL 값을 허용한다.

```
CREATE TABLE Recipient (
    recipient_id CHAR(10) NOT NULL,
    name VARCHAR(16) NULL,
    address VARCHAR(32) NOT NULL,
    contact VARCHAR(16) NULL,
    PRIMARY KEY (recipient_id)
);
```

Recipient의 Primary key인 recipient_id는 NULL이어서는 안되고, 고정 길이 10의 문자열로 정의한다.

이름, 주소, 연락처와 같은 값은 각 field에 맞는 충분한 길이의 가변 문자열로 정의하였다. 수령인의 주소 또한 없으면 배송을 할 수 없으므로 NULL이어서는 안된다.

```
CREATE TABLE Location (
    location_id CHAR(10) NOT NULL,
    location_name VARCHAR(16) NULL,
    location_type VARCHAR(16) NULL,
    location_contact VARCHAR(16) NULL,
    capacity INTEGER NULL,
    PRIMARY KEY (location_id)
);
```

Location의 Primary key인 location_id는 NULL이어서는 안되고, 고정 길이 10의 문자열로 정의한다. 이름, 타입, 연락처와 같은 값은 각 field에 맞는 충분한 길이의 가변 문자열로 정의하였다. capacity는 integer 값으로 각 위치의 용량을 수치화하여 저장한다.

```
CREATE TABLE Shipment (
    shipment_id CHAR(10) NOT NULL,
    shipment_enum INTEGER NOT NULL,
    timestamp DATETIME NULL,
    status VARCHAR(32) NULL,
```

```

        location_id CHAR(10) NOT NULL,
        PRIMARY KEY (shipment_id , shipment_enum)
    );
ALTER TABLE Shipment add constraint fk_shipment_location foreign key
(location_id) references Location(location_id) on delete cascade;

```

Shipment의 Primary key인 shipment_id, shipment_enum은 NULL이어서는 안되고, 고정 길이 10의 문자열로 정의하였다. Shipment_enum은 각 배송의 순서를 나타내는 값으로, 1, 2, 3...과 같은 integer로 설정하였으며 timestamp는 각 배송 단계의 시간을 나타내므로, 분 초까지 기입 가능한 DATETIME으로 설정하였다.

status는 정상 도착/전달, 사고 등의 상태를 기록하므로 32 길이로 충분하게 설정해 주었다. 각 배송은 어딘가의 위치에 속해 있어야 하므로 NOT NULL이다.

location_id는 location으로부터의 foreign key이며, 해당 장소가 사라지면 그 안의 배송품도 추적을 중지한다는 의미에서 on delete cascade를 설정하였다.

```

CREATE TABLE Package (
    package_id CHAR(10) NOT NULL,
    type VARCHAR(16) NULL,
    weight FLOAT NULL,
    service VARCHAR(16) NULL,
    ship_date DATETIME NULL,
    delivery_date DATETIME NULL,
    contents VARCHAR(16) NULL,
    customer_id CHAR(10) NULL,
    recipient_id CHAR(10) NULL,
    PRIMARY KEY (package_id)
);
ALTER TABLE Package add constraint fk_package_customer foreign key (customer_id) references Customer(customer_id) on delete cascade;
ALTER TABLE Package add constraint fk_package_recipient foreign key (recipient_id) references Recipient(recipient_id) on delete cascade;

```

Package의 Primary key인 package_id는 NULL이어서는 안되고, 고정 길이 10의 문자열로 정의하였다. type, service는 충분한 길이의 가변 문자열로 설정해주고, weight는 소수점 단위로 저장 가능해야 하므로 FLOAT로 설정하였다. 종류, 내용물, 발송자, 수신자는 대외비로 처리하는 문서 등을 위해 NULL 값을 허용하였다.

Customer로부터의 foreign key customer_id, Recipient로부터의 foreign key recipient_id는 해당 발송자 / 수취인의 정보가 사라지면 사용 내역도 삭제하는 개념으로 on delete cascade를 설정하였다. (회원 탈퇴 시 개인정보 파기)

```

CREATE TABLE Contain (
    package_id CHAR(10) NOT NULL,
    shipment_id CHAR(10) NOT NULL,
    shipment_enum INTEGER NOT NULL,
    PRIMARY KEY (package_id , shipment_id , shipment_enum)
);
ALTER TABLE Contain add constraint fk_contain_package foreign key (package_id) references Package(package_id) on delete cascade;
ALTER TABLE Contain add constraint fk_contain_shipment foreign key (s

```

```
hipment_id, shipment_enum) references Shipment(shipment_id, shipment_
enum) on delete cascade;
```

Contain의 Primary key인 package_id, shipment_id, shipment_enum은 NOT NULL로 설정해줬고, 각 foreign key 속성에서 참조 relation에 대해 on delete cascade를 해 줬는데, 이는 해당 배송 자체가 삭제되거나/배송품이 삭제되면 어떤 배송이 어떤 배송품을 포함하는 관계도 성립하지 않기 때문이다.

```
CREATE TABLE Billing (
    billing_id CHAR(10) NOT NULL,
    amount DECIMAL(19, 4) NULL,
    billing_date DATETIME NULL,
    billing_type CHAR(10) NULL,
    customer_id CHAR(10) NULL,
    package_id CHAR(10) NULL,
    PRIMARY KEY (billing_id)
);
ALTER TABLE Billing add constraint fk_billing_customer foreign key (c
ustomer_id) references Customer (customer_id) on delete cascade;
ALTER TABLE Billing add constraint fk_billing_package foreign key (pa
ckage_id) references Package (package_id) on delete cascade;
```

Billing의 Primary key인 billing_id는 NOT NULL로 설정해졌다. 가격을 나타내는 amount는 소수점 아래 4째자리까지 세밀하게 기입 가능하게 해 주었는데, cent와 같은 화폐 단위를 나타내거나, 아니면 기입/상태 유지 상의 편의를 위해 화폐 단위를 줄이는 경우 등의 확장성을 고려한 설정이다. (ex. 4500원 → 4.5)

primary key를 제외한 값들은 NULL 값 허용인데, 이는 아직 비용을 지불하지 않았거나, 선불인 경우 수취인/배송품 정보가 fix되지 않은 경우 등이 있을 수 있기 때문이다.

foeign key customer_id는 on delete cascade로 설정했는데, 이는 Customer relation 설명에 상술하였듯이 정보 제거의 이유이다. package_id는 배송품의 정보를 제거할 때, 연결된 지불 정보도 삭제하는 의미에서 on delete cascade로 설정하였다.

4. Queries & Code Implementation

```
Query Type : 0
=====
===== File loading...(drop) =====
#####
===== Table drop finished =====
```

```

printf("Connection Succeed\n");

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

////////// FILE INPUT ///////////
FILE* fp = fopen("20191559.txt", "r");
printf("\n===== File loading ... =====\n\n");
if (fp == NULL) {
    printf("File Open Error\n");
    return 1;
}

// input file with every line until EOF
char line[300];
while(fgets(line, 300, fp) != NULL) {
    //printf("%s", line);
    cout << "#";
    mysql_query(connection, line);
}
fclose(fp);

printf("\n===== File load finished =====\n\n");

```

20191559.txt의 내용을 읽어와 CRUD를 진행한다. 모든 줄을 읽고, sql 통신을 마치면 사용자 프롬프트를 본격적으로 시작한다. 아래는 20191559.txt의 일부분이다.

```

CREATE TABLE Customer(customer_id char(10) NOT NULL, name VARCHAR(16) NULL, address VARCHAR(32) NULL,
contact VARCHAR(16) NULL, account_number VARCHAR(32) NULL, PRIMARY KEY (customer_id));
CREATE TABLE Recipient( recipient_id char(10) NOT NULL, name VARCHAR(16) NULL, address VARCHAR(32) NOT
NULL, contact VARCHAR(16) NULL, PRIMARY KEY (recipient_id));
CREATE TABLE Location( location_id char(10) NOT NULL, location_name VARCHAR(16) NULL, location_type
VARCHAR(16) NULL, location_contact VARCHAR(16) NULL, capacity integer NULL, PRIMARY KEY (location_id));
CREATE TABLE Shipment( shipment_id char(10) NOT NULL, shipment_enum integer NOT NULL, timestamp DATETIME
NULL, status VARCHAR(32) NULL, location_id char(10) NOT NULL, PRIMARY KEY (shipment_id, shipment_enum));
ALTER TABLE Shipment add constraint fk_shipment_location foreign key (location_id) references
Location(location_id) on delete cascade;
CREATE TABLE Package( package_id char(10) NOT NULL, type VARCHAR(16) NULL, weight float NULL, service
VARCHAR(16) NULL, ship_date DATETIME NULL, delivery_date DATETIME NULL, contents VARCHAR(16) NULL,
customer_id char(10) NULL, recipient_id char(10) NULL, PRIMARY KEY (package_id));
ALTER TABLE Package add constraint fk_package_customer foreign key (customer_id) references
Customer(customer_id) on delete cascade;
ALTER TABLE Package add constraint fk_package_recipient foreign key (recipient_id) references
Recipient(recipient_id) on delete cascade;
CREATE TABLE Contain ( package_id char(10) NOT NULL, shipment_id char(10) NOT NULL, shipment_enum
integer NOT NULL, PRIMARY KEY (package_id, shipment_id, shipment_enum));
ALTER TABLE Contain add constraint fk_contain_package foreign key (package_id) references
Package(package_id) on delete cascade;
ALTER TABLE Contain add constraint fk_contain_shipment foreign key (shipment_id, shipment_enum)
references Shipment(shipment_id, shipment_enum) on delete cascade;
CREATE TABLE Billing( billing_id char(10) NOT NULL, amount DECIMAL(19,4) NULL, billing_date DATETIME
NULL, billing_type char(10) NULL, customer_id char(10) NULL, package_id char(10) NULL, PRIMARY KEY
(billing_id));
ALTER TABLE Billing add constraint fk_billing_customer foreign key (customer_id) references Customer
(customer_id) on delete cascade;
ALTER TABLE Billing add constraint fk_billing_package foreign key (package_id) references Package
(package_id) on delete cascade;
insert into Customer values ('C0001', 'SangWon Kang', 'Songpa, Seoul', '010-9912-8048', '8311-1010-1010');
insert into Customer values ('C0002', 'Rangho Lee', 'Mapo, Seoul', '010-7876-1234', '7711-1010-1010');
insert into Customer values ('C0003', 'Donald Trump', 'Washington', '010-2212-8228', '5511-1010-1010').

```

1-1. Find all customers who had a package on the truck at the time of the crash.

Query Type : 1
Truck number: 1
----- Subtypes in TYPE I -----
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.
Subtype : 1
---- TYPE I-1 ----
** Find all customers who had a package on the truck at the time of the crash. **
+-----+-----+-----+-----+-----+
customer_id name address contact account_number
+-----+-----+-----+-----+-----+
C0001 SangWon Kang Songpa, Seoul 010-9912-8048 8311-1010-1010
+-----+-----+-----+-----+-----+
C0002 Rangho Lee Mapo, Seoul 010-7876-1234 7711-1010-1010
+-----+-----+-----+-----+-----+
C0003 Donald Trump Washington 010-2212-8228 5511-1010-1010
+-----+-----+-----+-----+-----+

```

if (!type) break;
else if (type==1) {
    printf("\n---- TYPE I-1 ----\n");
    printf("** Find all customers who had a package on the truck at the time of the
crash. **\n");

    const char* query =
        "SELECT DISTINCT c.* \
        FROM Customer c \
        JOIN Package p ON c.customer_id = p.customer_id \
        JOIN Contain cn ON p.package_id = cn.package_id \
        JOIN Shipment s ON cn.shipment_id = s.shipment_id AND cn.shipment_enum = 6 \
        JOIN Location l ON s.location_id = l.location_id \
        WHERE l.location_type = 'truck' AND l.location_id = 'LT0001';";

    query_print(query);
}

```

Truck 1 (LT001)이 Shipment 도중 6번째에서 Crash가 남아 기록되어 있다. 해당 시점에 트럭에 있었던 배송품들의 정보를 얻어야 하므로, Package, Contain을 Join하여 해당 shipment 단계에 있었던 배송품들을 확인하고 location이 해당 트럭, 해당 shipment 단계임을 확인한다. 배송 사고가 난 배송품들의 고객 정보가 필요하므로 distinct한 Package의 customer_id 값과 더불어 이름, 연락처 등을 표시한다. 해당 정보를 통해 연락을 취하거나 refund 등을 진행할 수 있을 것이다.

1-2. Find all recipients who had a package on that truck at the time of the crash.

```
----- Subtypes in TYPE I -----
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.

Subtype : 2

---- TYPE I-2 ----
** Find all recipients who had a package on that truck at the time of the crash. **
+-----+
| recipient_id | name        | address      | contact     |
+-----+
| R0001         | Yeonghee    | Busan        | 010-2349-8432 |
| R0002         | Cheolsu     | Daejeon     | 010-3432-5423 |
| R0003         | Gicheol     | Daegu        | 010-2309-2435 |
+-----+



else if (type==2) {
    printf("\n---- TYPE I-2 ----\n");
    printf("** Find all recipients who had a package on that truck at the time of the
crash. **\n");

    const char* query =
        "SELECT DISTINCT r.* \
        FROM Recipient r \
        JOIN Package p ON r.recipient_id = p.recipient_id \
        JOIN Contain cn ON p.package_id = cn.package_id \
        JOIN Shipment s ON cn.shipment_id = s.shipment_id AND cn.shipment_enum = 6 \
        JOIN Location l ON s.location_id = l.location_id \
        WHERE l.location_type = 'truck' AND l.location_id = 'LT0001';";
    query_print(query);
}
```

Type I-2와 마찬가지로 해당 사고 시점의 트럭에 있었던 배송품들을 조회하고, 그 배송품들의 수신자를 distinct하게 출력한다.

1-3. Find the last successful delivery by that truck prior to the crash.

```
----- Subtypes in TYPE I -----
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.

Subtype : 3

---- TYPE I-3 ----
** Find the last successful delivery by that truck prior to the crash. **
+-----+
| shipment_id | shipment_enum | timestamp      | status        | location_id   |
+-----+
| S0001       | 4             | 2023-01-24 21:30:20 | move to DongSeoul Center | LT0001        |
+-----+
```

```

else if (type == 3) {
    printf("\n---- TYPE I-3 ----\n");
    printf("** Find the last successful delivery by that truck prior to the crash.
**\n");

    const char* query =
        "SELECT s1.* \
         FROM Shipment s1 \
         JOIN Location l1 ON s1.location_id = l1.location_id \
         WHERE l1.location_type = 'truck' AND l1.location_id = 'LT0001' \
         AND s1.timestamp < (SELECT s2.timestamp FROM Shipment s2 WHERE s2.status = \
         'Crash') \
         ORDER BY s1.timestamp desc \
         LIMIT 1;";
    query_print(query);
}

```

Crash 시점은 shipment status에 ‘Crash’로 기입되어 있는 시점일 것이다. 따라서 timestamp 값이 사고 시점 값보다 작고, 해당 트럭이 배송한 마지막 배송을 찾는다. 내림차순으로 정렬하고, LIMIT 1로 최대값을 표출한다.

2. Find the customer who has shipped the most packages in the past year.

```

Query Type : 2
:
---- TYPE II ----
** Find the customer who has shipped the most packages in the past year. ** (Hint: 2023)
Year: 2023
+-----+
| customer_id | name      | address     | contact      | account_number | package_count |
+-----+
| C0003       | Donald Trump | Washington | 010-2212-8228 | 5511-1010-1010 | 2             |
+-----+

else if (type==2) {
    printf("\n\n---- TYPE II ----\n");
    printf("** Find the customer who has shipped the most packages in the past year. ** (Hint:
2023)\n");
    int year;
    cout << "\nYear: ";
    cin >> year;
    char query[500];
    sprintf(query,
        "SELECT c.*, COUNT(p.package_id) AS package_count \
         FROM Customer c \
         JOIN Package p ON c.customer_id = p.customer_id \
         WHERE YEAR(p.ship_date) = %d \
         GROUP BY c.customer_id \
         ORDER BY package_count DESC \
         LIMIT 1;", year);
    query_print(query);
}

```

사용자에게 프롬포트로 입력받은 년도(예: 2023)를 query 문 안에 넣어서 처리한다. package의 ship_date가 해당 년도이고, 이를 customer_id로 묶은 다음 package들의 개수를 count하여

정렬 후 최대값을 찾는다.

3. Find the customer who has spent the most money on shipping in the past year.

```
Query Type : 3
---- TYPE III ----
** Find the customer who has spent the most money on shipping in the past year. ** (Hint: 2023)
Year: 2023
+-----+
| customer_id | name        | total_amount |
+-----+
| C0001       | SangWon Kang | 22000.0000   |
+-----+


else if (type==3) {
    printf("\n---- TYPE III ----\n");
    printf("** Find the customer who has spent the most money on shipping in the past year. **\n(Hint: 2023)\n");
    int year;
    cout << "\nYear: ";
    cin >> year;
    char query[500];
    sprintf(query,
            "SELECT b.customer_id, c.name, SUM(b.amount) as total_amount \
             FROM Billing b \
             JOIN Customer c ON b.customer_id = c.customer_id \
             WHERE YEAR(b.billing_date) = %d \
             GROUP BY b.customer_id, c.name \
             ORDER BY total_amount DESC \
             LIMIT 1;", year);
    query_print(query);
}
```

사용자에게 프롬프트로 입력받은 년도에 있었던 billing들을 모두 찾은 후에, amount 값들의 sum이 가장 큰 사용자를 찾는다. 총 금액을 total_amount로서 출력하고, customer_id, customer_name도 출력하여 최우수 고객의 정보를 표출한다.

4. Find the packages that were not delivered within the promised time.

```
Query Type : 4
---- TYPE IV ----
** Find the packages that were not delivered within the promised time. **
+-----+
| package_id | promised_time | shipment_id | actual_delivery |
+-----+
| PK0003     | 2023-05-30 14:50:30 | S0003      | 2023-06-12 22:34:20 |
+-----+
```

```

else if(type==4) {
    printf("\n---- TYPE IV ----\n");
    printf("** Find the packages that were not delivered within the promised time. **\n");
    const char* query =
        "SELECT P.package_id, P.delivery_date as promised_time, S.shipment_id, S.timestamp as
         actual_delivery \
        FROM Package P \
        JOIN Contain C ON P.package_id = C.package_id \
        JOIN Shipment S ON C.shipment_id = S.shipment_id AND C.shipment_enum = S.shipment_enum
        WHERE S.status = 'delivered' AND S.timestamp > P.delivery_date;";

    query_print(query);
}

```

Package에 기입되어 있는 delivery_date(약속 기한)과 Shipment에 기입되어 있는 마지막 단계 ‘delivered’의 시점을 비교한다. 실제 배송 완료 시점이 약속 기한보다 나중인 shipment를 찾고, 이 안에 있었던 package들을 표출한다.

5. Generate the bill for each customer for the past month.

I. A simple bill: customer, address, and amount owed.

```

Query Type : 5
-----
----- TYPE V -----
** Generate the bill for each customer for the past month. **

Year Month (hint: 2019 1): 2019 1

Which type of bill do you want?
    1. A simple bill: customer, address, and amount owed.
    2. A bill listing charges by type of service.
    3. An itemize billing listing each individual shipment and the charges for it.

Bill type: 1
+-----+
| customer_id | name       | address      | total_amount |
+-----+
| C0001       | SangWon Kang | Songpa, Seoul | 15000.0000   |
| C0002       | Rangho Lee   | Mapo, Seoul   | 70000.0000   |
| C0003       | Donald Trump  | Washington   | 114000.0000  |
| C0004       | Putin        | Moscow        | 47000.0000   |
| C0005       | Xi Jinping   | Beijing       | 20000.0000   |
+-----+

case 1:
    sprintf(query,
        "SELECT b.customer_id, c.name, c.address, SUM(b.amount) as total_amount \
        FROM Billing b \
        JOIN Customer c ON b.customer_id = c.customer_id \
        WHERE YEAR(b.billing_date) = %d AND MONTH(b.billing_date) = %d \
        GROUP BY b.customer_id, c.name, c.address;", year, month);
    query_print(query);
    break;

```

고객, 주소, 총 사용 금액을 표출한다. 사용자에게 프롬포트로 받은 년도에 해당하는 Billing을 확인하고, 이에 연계되는 배송품들과 해당 고객의 정보를 찾는다.

II. A bill listing charges by type of service.

```
---- TYPE V ----
** Generate the bill for each customer for the past month. **

Year Month (hint: 2019 1): 2019 1

Which type of bill do you want?
  1. A simple bill: customer, address, and amount owed.
  2. A bill listing charges by type of service.
  3. An itemize billing listing each individual shipment and the charges for it.

Bill type: 2
+-----+
| customer_id | name       | service    | total_amount |
+-----+
| C0001       | SangWon Kang | slow       | 15000.0000   |
| C0002       | Rangho Lee   | slow       | 30000.0000   |
| C0002       | Rangho Lee   | fast       | 40000.0000   |
| C0003       | Donald Trump | fast       | 114000.0000  |
| C0004       | Putin        | fast       | 47000.0000   |
| C0005       | Xi Jinping   | fast       | 20000.0000   |
+-----+
```

case 2:

```
sprintf(query,
    "SELECT b.customer_id, c.name, p.service, SUM(b.amount) as total_amount \
     FROM Billing b \
     JOIN Customer c ON b.customer_id = c.customer_id \
     JOIN Package p ON b.package_id = p.package_id \
     WHERE YEAR(b.billing_date) = %d AND MONTH(b.billing_date) = %d \
     GROUP BY b.customer_id, c.name, p.service;", year, month);
query_print(query);
```

각 사용자별로 서비스 타입에 따른 총 가격들을 표시한다. Group by로 각 customer, 그리고 그 각각에 대해 서비스별로 정렬하여 출력한다. 위 예시에 보면 ‘Rangho Lee’의 ‘slow’ 서비스의 총 가격이 30000, ‘fast’ 서비스가 40000이다.

III. An itemized billing listing each individual shipment and the charges for it.

```
Query Type : 5
---- TYPE V ----
** Generate the bill for each customer for the past month. **

Year Month (hint: 2019 1): 2019 1

Which type of bill do you want?
  1. A simple bill: customer, address, and amount owed.
  2. A bill listing charges by type of service.
  3. An itemize billing listing each individual shipment and the charges for it.

Bill type: 3
+-----+
| billing_id | name       | package_id | service    | amount   |
+-----+
| B0001       | SangWon Kang | PK0001    | slow       | 15000.0000 |
| B0003       | Rangho Lee   | PK0003    | slow       | 30000.0000 |
| B0004       | Rangho Lee   | PK0004    | fast       | 40000.0000 |
| B0005       | Donald Trump | PK0005    | fast       | 38000.0000 |
| B0006       | Donald Trump | PK0006    | fast       | 42000.0000 |
| B0007       | Donald Trump | PK0007    | fast       | 34000.0000 |
| B0008       | Putin        | PK0008    | fast       | 37000.0000 |
| B0009       | Putin        | PK0009    | fast       | 10000.0000 |
| B0010       | Xi Jinping   | PK0010    | fast       | 20000.0000 |
+-----+
```

```

case 3:
    sprintf(query,
        "SELECT b.billing_id, c.name, b.package_id, p.service, b.amount \
        FROM Billing b \
        JOIN Customer c ON b.customer_id = c.customer_id \
        JOIN Package p ON b.package_id = p.package_id \
        WHERE YEAR(b.billing_date) = %d AND MONTH(b.billing_date) = %d;", year, month);
    query_print(query);
    break;
}

```

각 독립적인 Shipment에 대해 가격과 bill 정보를 출력한다. 사용자에게 프롬포트로 입력 받은 년도와 월(ex. 2019 1)을 기반으로 해당 기간에 있었던 배송들의 정보를 찾는다.

```

Query Type : 0
===== File loading...(drop) =====
#####
===== Table drop finished =====

////////// FILE INPUT ///////////
FILE* fp2 = fopen("20191559_drop.txt", "r");
printf("\n===== File loading ... (drop) =====\n\n");
if (fp2 == NULL) {
    printf("File Open Error\n");
    return 1;
}
// input file with every line until EOF
char line[300];
while (fgets(line, 300, fp2) != NULL) {
    cout << "#";
    mysql_query(connection, line);
}
printf("\n\n===== Table drop finished =====\n");
fclose(fp2);

mysql_close(connection);

return 0;

```

마지막으로 사용자가 최상위 query select 메뉴에서 0을 입력하여 모든 query execution이 끝나면 2019155_drop.txt에 있는 drop문들을 실행하여 DB를 비워준다.

```

drop table if exists Shipment;
drop table if exists Billing;
drop table if exists Package;
drop table if exists Customer;
drop table if exists Recipient;
drop table if exists Location;

```