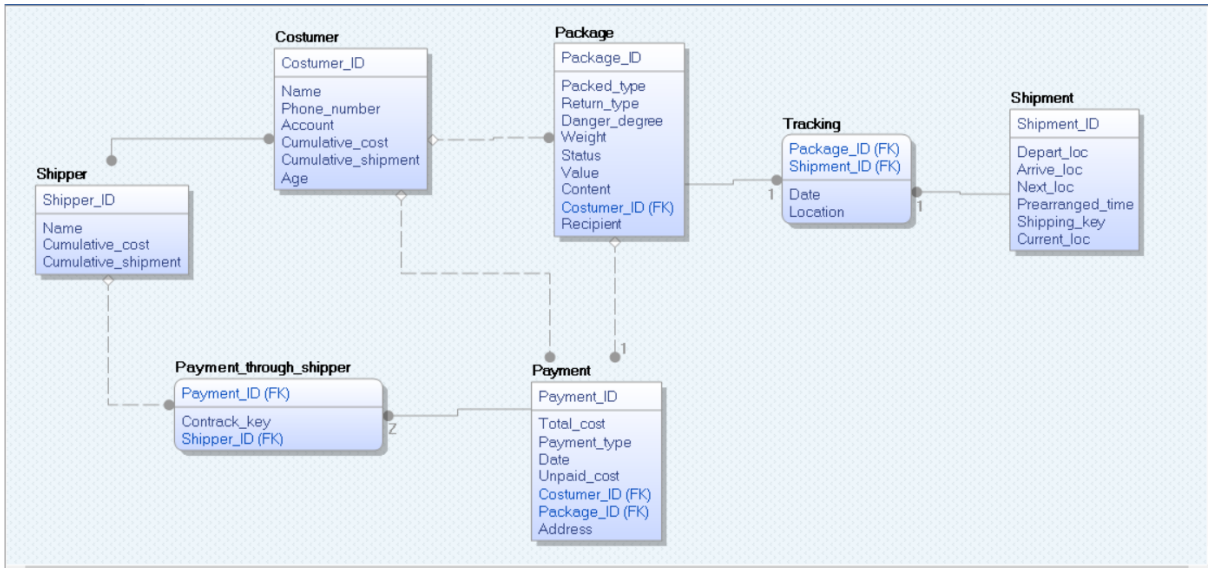
DBMS Project 2 Report

20180492 김범준

1. BCNF Decomposition



위 그림은 project 1에서 완성된 E-R diagram이다. 각 entity별로 Functional Dependency를 확인하여, BCNF의 조건을 위배할 경우 decomposition을 진행하였다.

- Costumer

Costumer entity에 존재하는 functional dependency는 다음과 같다.

- Costumer_ID -> Name, Phone_number, Account, Cumulative_cost, Cumulative_shipment, Age

이외에 functional dependency는 없다고 판단하였다. Name attribute의 경우 동명이인이 있을 수 있기 때문에 다른 attribute로 가는 dependency가 없고, Phone_number의 경우 어린이 회원과 같이 전화번호가 없는 고객의 경우 다른 전화번호를 사용할 수 있기 때문에 다른 attribute로 가는 dependency가 없다. Account의 경우에도 앞선 예시와 같이, 본인의 명의가 아닌 계좌번호를 사용

할 수 있기 때문에 다른 attribute로 가는 dependency가 없다. 유일한 functional dependency의 경우 Costumer_ID가 PK이기 때문에 BCNF의 조건을 만족한다. 따라서 별도의 decomposition을 진행하지 않았다.

- Package

Package entity에 존재하는 functional dependency는 다음과 같다.

- Package_ID -> Packed_type, Return_type, Danger_degree, Weight, Status, Value, Content, Recipient, Costumer_ID

이외에 functional dependency는 없다고 판단하였다. FK인 Costumer_ID의 경우 한 고객이 여러 package를 배송주문했을 수 있으므로 다른 attribute로 가는 dependency를 가지지 않는다. 유일한 functional dependency의 경우, Package_ID가 PK에 해당하므로, BCNF의 조건을 만족한다. 따라서 Package entity에서는 decomposition을 하지 않았다.

- Payment

Payment entity에 존재하는 functional dependency는 다음과 같다.

- Payment_ID -> Total_cost, Payment_type, Date, Unpaid_cost, Address, Package_ID
- Package_ID -> Total_cost, Payment_type, Date, Unpaid_cost, Address

이외에 functional dependency는 없다고 판단하였다. Address의 경우 같은 고객의 다른 배송, 혹은 다른 고객의 배송이 같은 주소지로 배송이 요청될 수 있기 때문에 다른 attribute로 가는 dependency가 없다. 첫 번째 functional dependency의 경우 Payment_ID가 PK이기 때문에 BNCF의 조건을 만족한다. 두 번째 functional dependency의 경우 FK인 Package_ID가 Candidate key로, SK에 해당하기 때문에 BCNF의 조건을 만족한다. 따라서 별도의 decomposition을 진행하지 않았다.

- Shipper

Shipper entity에 존재하는 functional dependency는 다음과 같다.

- Shipper_ID -> Name, Cummulative_cost, Cummulative_shipment

이외에 functional dependency는 없다고 판단하였다. Name의 경우, 흔치 않은 경우지만 다른 배송업체의 이름이 같을 수 있기 때문에 다른 attribute로 가는 dependency가 없다. 유일한

functional dependency의 경우 Shipper_ID가 PK이기 때문에 BCNF의 조건을 만족한다. 따라서 별도의 decomposition을 진행하지 않았다.

- Shipment

먼저, 기존에 shipment에 존재하던 attribute인 shipping_key를 tracking entity로 이동시켰다. tracking entity는 배송물이 어느 위치에 있는지를 담고 있는데, 위치를 이동할 때 마다 이동수단이 달라질 수 있기 때문에 tracking entity가 shipping_key를 가지고 있는 것이 적절하다고 판단하였다.

Shipment entity에 존재하는 functional dependency는 다음과 같다.

- Shipment_ID -> Depart_loc, Arrive_loc, Next_loc, Current_loc, Prearranged_time

이외의 functional dependency는 없다고 판단하였다. 배송 자체의 고유의 식별 가능한 ID 이외의 다른 attribute는 별다른 dependency를 가지지 않는다. 유일한 functional dependency의 경우 Shipment_ID가 PK이기 때문에 BCNF의 조건을 만족한다. 따라서 별도의 decomposition을 진행하지 않았다.

- Tracking

앞서 shipment entity에 대한 설명에서 언급한 것처럼, 기존에 shipment가 가지고 있던 attribute인 shipping_key를 tracking으로 옮겨왔다.

Tracking entity는 relation attribute를 가지는 relation을 entity로 확장한 것이다. Tracking entity에 존재하는 functional dependency는 다음과 같다.

- Package_ID -> Shipment_ID, Date, Location, tracking
- Shipment_ID -> Package_ID, Date, Location, tracking

다음 functional dependency를 보면 알 수 있듯이, 기존의 PK로 설정하였던 (Package_ID, Shipment_ID)를 하나씩 나누어 Package_ID 혹은 Shipment_ID 하나만을 PK로 사용하여도 문제가 없다. 즉, Package_ID와 Shipment_ID가 candidate key가 되며 (Package_ID, Shipment_ID)는 PK가 될 수 없다. 따라서 Package_ID를 유일한 PK로 사용하고, Shipment_ID를 PK가 아닌 attribute로 변경하였다. 해당 변경사항은 뒤의 Physical diagram에 반영하도록 하였다. 두 개의 functional dependency의 경우 Package_ID는 PK, Shipment_ID는 candidate key이므로 BCNF의 조건을 모두 만족한다. 따라서 PK를 수정한 것 이외의 별도의 decomposition을 진행하지 않았다.

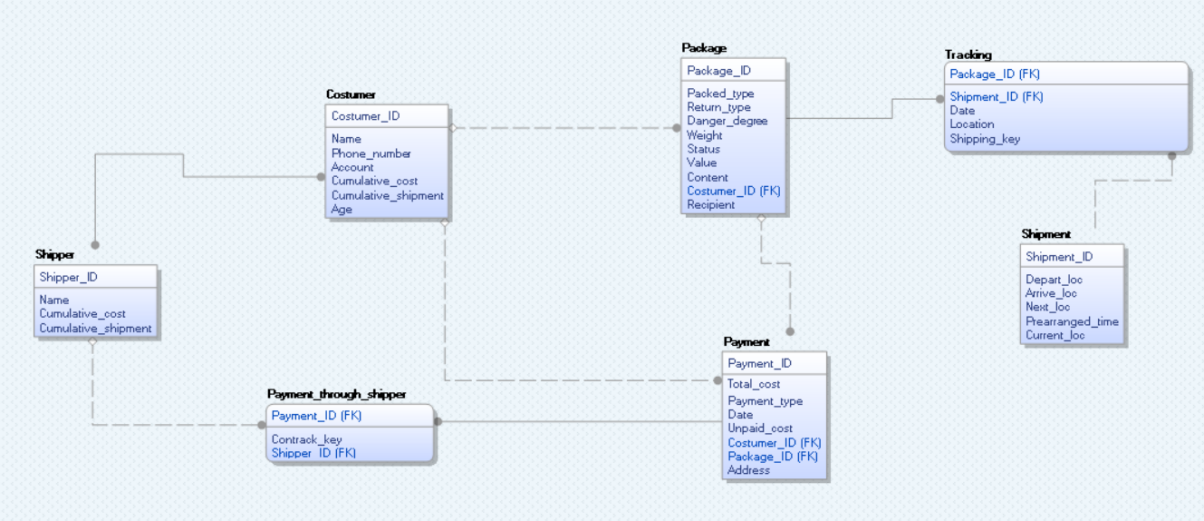
- Payment_Through_Shipper

Payment_Through_Shipper는 relation attribute를 가지는 relation을 entity로 확장한 것이다. Payment_Through_Shipper에 존재하는 functional dependency는 다음과 같다.

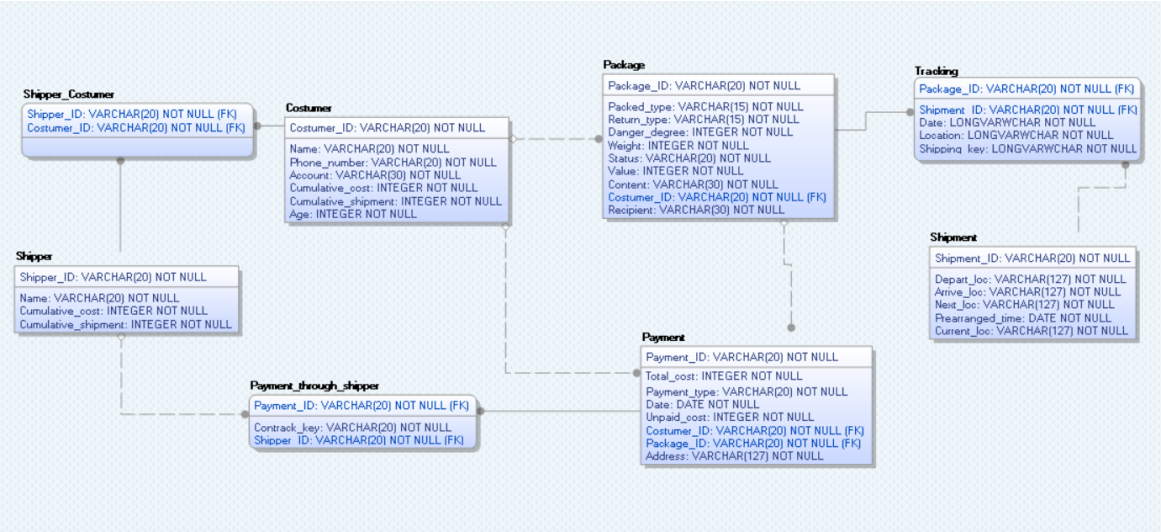
- Payment_ID -> Contract_key, Shipper_ID

이외의 functional dependency는 없다고 판단하였다. FK인 Shipper_ID의 경우 한 배송업체가 여러 건의 지불을 관리할 수 있으므로, 다른 attribute로 가는 dependency가 존재하지 않는다. 유일한 functional dependency의 경우 Payment_ID가 PK이므로 BCNF의 조건을 만족한다. 따라서 별도의 decomposition을 진행하지 않았다.

결론적으로, Tracking entity에서 PK를 수정한 것과 Shipping_key attribute를 shipment entity에서 Tracking entity로 옮긴 것 이외의 다른 decomposition은 필요하지 않다고 판단하였다. 최종적으로 수정된 logical E-R diagram은 다음과 같다.



2. Physical Schema Diagram



다음은 1번에서 수정된 logical schema diagram을 physical schema diagram으로 변환한 것이다. physical로의 전환 시 Shipper와 Costumer의 many-to-many relationship인 Shipper_Costumer가 entity로 추가되었다. 각 entity에 대한 설명은 아래와 같다.

- Costumer

- Costumer_ID : Costumer_ID는 숫자 n자리(e.g. 15567829304) 혹은 문자와 숫자의 조합(e.g. AGKD100983)으로 이루어질 것을 고려하여 varchar(20) type으로 정하였다. 또한 Costumer_ID는 PK이므로 not null이다.

- Name : Name은 문자로 이루어져 있고, 사람마다 가질 수 있는 이름의 길이가 다르므로 varchar(20)으로 정하였다. 또한, 이름이 없는 고객은 있을 수 없으므로 not null이다.

- Phone_number : 전화번호는 숫자와 하이픈(e.g. 010-1234-5678)으로 이루어져 있고 여러 종류의 전화번호 및 길이가 존재할 수 있으므로 넉넉하게 varchar(20)으로 정하였다. 전화번호, 즉 연락수단이 존재하지 않는 고객은 존재하지 않으므로 not null이다.

- Account : Account는 문자, 언더바 혹은 하이픈(_ , -)으로 이루어진다.(e.g. Kookmin_123456_78_901234) 또한 은행마다 계좌번호의 길이가 다르다. 따라서 varchar(30)으로 조금 넉넉하게 정하였다. 또한 계좌번호가 없는 고객은 거래를 할 수 없으므로 not null이다.

- Cummulative_cost : Cummulative_cost는 누적금액을 나타내는 attribute이므로 integer로 정하였다. 또한 누적금액이 존재하지 않는 고객은 0의 값을 가지고, 값 자체가 존재하지 않는 고객은 없으므로 not null이다.
- Cummulative_shipment : Cummulative_shipment는 Cummulative_cost와 비슷하게 누적배송주문수를 나타내는 attribute이므로 integer로 정하였다. 또한 Cummulative_shipment도 값 자체가 존재하지 않는 고객은 없으므로 not null이다.
- Age : Age는 나이를 나타내므로 integer로 정하였다. 또한 나이가 존재하지 않는 고객은 없으므로 not null이다.

- Package

- Package_ID : Package_ID는 Costumer_ID와 같이 숫자 n자리, 문자와 숫자의 조합으로 구성되는 경우를 생각하여 varchar(20)으로 설정하였다. 또한 PK이므로 not null이다.
- Packed_type : Packed_type은 배송물이 박스, 봉투와 같이 어떠한 방식으로 포장되었는지 나타내는 attribute이다. box_large와 같이 크기와 함께 저장하므로 varchar(20)으로 정하였다. 포장이 되지 않은 배송물은 없으므로 not null이다.
- Return_type : Return_type은 배송물이 일반배송인지 반품배송인지 나타내는 attribute이고 varchar(20)으로 설정하였다. 일반배송, 반품배송이 아닌 배송물은 존재하지 않으므로 not null이다.
- Danger_degree : Danger_degree는 배송물의 위험도를 나타내는 attribute이고 1등급, 2등급과 같이 숫자로 표현되므로 integer로 정하였다. 위험도가 존재하지 않는 배송물은 없으므로 not null이다.
- Weight : Weight는 배송물의 무게를 나타내며, integer로 정하였다. 무게가 없는 배송물은 존재하지 않으므로 not null이다.
- Status : Status는 현재 배송물의 상태(e.g. good, broken)를 나타내는 attribute이다. 배송물의 상태는 문자열로 나타나므로 varchar(20)으로 정하였다. 현재 상태가 존재하지 않는 배송물은 없으므로 not null이다.
- Value : Value는 배송물의 가치를 현금으로 환산한 값을 나타내며 integer로 나타내었다. Value가 존재하지 않는 배송물은 없으므로 not null이다.
- Content : Content는 배송물의 내용물을 나타낸다. 내용물은 television, laptop과 같이 문자열로 나타내므로 varchar(30)으로 정하였다. 내용물이 없는 배송물은 존재하지 않으므로 not null이다.

- Costumer_ID : Costumer의 PK이며 Package의 FK이다. data type과 not null은 Costumer entity의 Costumer_ID와 동일하다.
- Recipient : Recipient는 수취인이 누군지 나타낸다. 수취인의 이름은 문자열이므로 varchar(30)으로 설정하였다. 수취인이 없는 배송물은 존재하지 않으므로 not null이다.

- Payment

- Payment_ID : Payment_ID는 Costumer_ID와 같이 숫자 n자리, 문자와 숫자의 조합으로 구성되는 경우를 생각하여 varchar(20)으로 설정하였다. 또한 PK이므로 not null이다.
- Total_cost : Total_cost는 한 배송건에 관하여 고객이 지불해야 할 금액을 나타내며 integer로 나타내었다. 지불해야 하는 금액이 존재하지 않는 배송은 존재하지 않으므로 not null이다.
- Payment_type : Payment_type은 고객의 지불 방법을 나타낸다. credit_card, account_transfer와 같이 나타내므로 varchar(20)로 정하였다. 지불 방법이 존재하지 않는 결제는 없으므로 not null이다.
- Date : Date는 결제 일자를 나타내며 date data type을 사용하였다. 결제 일자가 없는 결제는 존재하지 않으므로 not null이다.
- Unpaid_cost : Unpaid_cost는 총 금액 중 고객이 지불하지 않은 금액을 나타내며 값의 범위는 0 ~ Total_cost이다. Unpaid_cost가 없는 결제는 없으므로 not null이다.
- Costumer_ID, Package_ID : 각각 Costumer, Package entity의 PK이며 Payment의 FK이다. data type과 not null은 각 entity의 것을 따른다.
- Address : Address는 지불이 요청되는 주소를 나타낸다. 11-11, sinsu-dong, mapo-gu, seoul, Republic of Korea와 같이 나타내므로 넉넉하게 varchar(127)로 정하였다. 주소가 없는 결제는 존재하지 않으므로 not null이다.

- Shipper

- Shipper_ID : Shipper_ID는 Costumer_ID와 같이 숫자 n자리, 문자와 숫자의 조합으로 구성되는 경우를 생각하여 varchar(20)으로 설정하였다. 또한 PK이므로 not null이다.
- Name : Name은 배송업체의 이름을 나타낸다. 이름은 문자열로 표현되므로 varchar(20)으로 정하였다. 이름이 없는 배송업체는 존재하지 않으므로 not null이다.
- Cummulative_cost : Cummulative_cost는 해당 배송업체에서 주문된 모든 결제의 지불금

액을 합한 것을 나타낸다. integer로 나타내었으며 누적주문금액이 없는 배송업체는 존재하지 않으므로 not null이다.

- Cummulative_shipment : Cummulative_shipment는 해당 배송업체의 누적 주문건수를 나타낸다. integer로 나타내었으며 누적주문건수가 없는 배송업체는 존재하지 않으므로 not null이다.

- Shipment

- Shipment_ID : Shipment_ID는 Costumer_ID와 같이 숫자 n자리, 문자와 숫자의 조합으로 구성되는 경우를 생각하여 varchar(20)으로 설정하였다. 또한 PK이므로 not null이다.

- Depart_loc : Depart_loc는 배송의 출발 지점을 나타낸다. 배송의 출발 지점은 okchun hub와 같이 문자열로 나타나므로 varchar(127)으로 정하였다. 출발 지점이 없는 배송은 존재하지 않으므로 not null이다.

- Arrive_loc : Arrive_loc는 배송의 도착 지점을 나타낸다. 배송의 도착 지점은 11-11, sinsu-dong, mapo-gu, seoul, Republic of Korea와 같이 표현 가능하므로 varchar(127)로 정하였다. 도착 지점이 없는 배송은 존재하지 않으므로 not null이다.

- Next_loc : Next_loc는 배송이 향하는 다음 지점을 나타낸다. okchun hub와 같이 나타낼 수 있으므로 varchar(127)로 정하였다. 배송이 최종 목적지에 도착했을 경우 none의 값을 가지도록 하므고, 항상 값을 가지고 있어야 한다. 따라서 not null이다.

- Prearranged_time : Prearranged_time은 배송의 정해진 도착 시간을 나타내며 date로 표현하였다. 정해진 도착 시간이 없는 배송은 존재하지 않으므로 not null이다.

- Current_loc : Current_loc는 배송물의 현재 위치를 나타낸다. okchun hub와 같이 나타낼 수 있으므로 varchar(127)로 정하였다. 배송물의 현재 위치가 존재하지 않는 경우는 없으므로 not null이다.

- Tracking

- Package_ID : Package entity의 PK이며 Tracking의 FK이자 PK이다. 따라서 Package의 PK인 Package_ID와 같은 data type을 가지며, not null이다.

- Shipment_ID : Shipment entity의 PK이며 Tracking의 FK이자 CK이다. 따라서 Shipment의 PK인 Shipment_ID와 같은 data type을 가지며, not null이다.

- Date : 해당 배송물이 이전의 각 위치에 언제 도착했는지 나타내는 attribute이다. logical

schema에서 여러 값들을 가지고 있어야 하므로 array data type이었고 physical로 전환 시 longvarwchar의 type을 가지게 되었다.

- Location : 해당 배송물이 이전에 어느 위치에 있었는지 나타내는 attribute이다. logical schema에서 여러 값들을 가지고 있어야 하므로 array data type이었고 physical로 전환 시 longvarwchar의 type을 가지게 되었다.
- Shipping_key : Shipping_key는 현재 배송중인 배송물의 배송 수단이 가지는 식별 가능한 ID이다. logical schema에서 여러 값들을 가지고 있어야 하므로 array data type이었고 physical로 전환 시 longvarwchar의 type을 가지게 되었다.

- Payment_through_shipper

- Payment_ID : Payment_ID는 Payment entity의 PK이자 Payment_through_shipper의 FK이자 PK이다. 따라서 Payment의 PK인 Payment_ID와 같은 data type을 가지며 not null이다.
- Contract_key : Contract_key는 해당 결제가 일시적인 결제인지, 혹은 구독 결제인지 나타낸다. subscripted과 같이 나타낼 수 있으므로 varchar(20)으로 정하였다. 배송 업체와 계약된 결제일 경우 무조건 존재하야 하므로 not null이다.
- Shipper_ID : Shipper_ID는 Shipper entity의 PK이자 Payment_through_shipper의 FK이다. 따라서 Shipper의 PK인 Shipper_ID와 같은 data type을 가지며 not null이다.

- Shipper_costumer

- Shipper_ID, Costumer_ID : 해당 attribute들은 각각 Shipper와 Costumer entity의 PK이며 Shipper_costumer의 PK이자 FK이다. 따라서 varchar(20)이며 not null이다.

3. Queries

1) TYPE I : Assume truck X(truck_1721) is destroyed in a crash (crash date : 2023-03-03)

query 1의 경우 사고가 난 트럭을 truck_1721, 사고 날짜를 2023년 3월 3일로 가정하였다.

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

```
select distinct P.customer_ID
from package P natural join tracking T
where T.shipping_key = 'truck_1721' and T.trackdate = '2023-03-03' and P.package_ID = T.package_ID order by P.customer_ID
```

사용한 sql문은 위와 같다. package entity와 customer entity를 join하여 배송물을 배송하던 수단이 truck_1721, 날짜가 2023년 3월 3일, package entity의 package_ID와 tracking entity의 package_ID가 같은 customer_ID를 추출하였다.

```
if (subcommand == 1) {
    printf("---- TYPE I-1 ----\n\n");
    printf("**Find all customers who had a package on the truck at the time of the crash**\n");
    vector<string> cid;
    string query = "select distinct P.customer_ID from package P natural join tracking T where T.shipping_key = 'truck_1721'
    int state = 0;
    state = mysql_query(connection, query.c_str());
    if (state == 0)
    {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            string tmp(sql_row[0]);
            cid.push_back(tmp);
        }
        mysql_free_result(sql_result);
    }
    if (!cid.size()) {
        printf("| none. |\n\n");
        continue;
    }
    for (int i = 0; i < cid.size(); i++) {
        string subquery = "select customer_ID, name from customer where customer_ID = '" + cid[i] + "'";
        state = 0;
        state = mysql_query(connection, subquery.c_str());
        if (state == 0)
        {
            sql_result = mysql_store_result(connection);
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                printf("| %s %s ", sql_row[0], sql_row[1]);
            }

            mysql_free_result(sql_result);

        }
    }
    printf("|\n\n");
```

또한 위의 코드와 같이 첫 번째 sql에서 추출된 customer_ID를 cid 벡터에 push_back하여 저장하고, cid의 길이만큼 루프를 돌며 추출된 customer_ID에 해당하는 이름을 찾아 customer_ID와 함께 출력하였다. 결과는 다음과 같다.

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

        1. TYPE I
        2. TYPE II
        3. TYPE III
        4. TYPE IV
        5. TYPE V
        0. QUIT
Select query type : 1
----- Subtypes in TYPE I -----
        1. TYPE I-1.
        2. TYPE I-2.
        3. TYPE I-3.
Select query type of TYPE I : 1
---- TYPE I-1 ----

**Find all customers who had a package on the truck at the time of the crash**
| A1234567 chulsoo kim | A3456789 hun jegal |
```

CRUD 텍스트에서 chulsoo kim과 hun jegal의 배송물이 2023년 3월 3일에 truck_1721로 배송중이었고, 해당 결과가 출력되었다.

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

```
select recipient
from package natural join tracking
where shipping_key = 'truck_1721' and trackdate = '2023-03-03'
```

사용한 sql문은 위와 같다. package entity와 tracking entity를 natural join하여 2023년 3월 3일 truck_1721을 통해 배송중이던 배송물의 수취인을 추출하였다. 결과는 다음과 같다.

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

        1. TYPE I
        2. TYPE II
        3. TYPE III
        4. TYPE IV
        5. TYPE V
        0. QUIT
Select query type : 1
----- Subtypes in TYPE I -----
        1. TYPE I-1.
        2. TYPE I-2.
        3. TYPE I-3.
Select query type of TYPE I : 2
---- TYPE I-2 ----

**Find all recipients who had a package on the truck at the time of the crash**
| yujin kim | younghee lee |
```

해당 수취인은 truck_1721이 파손될 때 들어있던 배송물의 수취인이다.

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

```
select shipment_ID, trackdate

from tracking

where shipping_key = 'truck_1721' and trackdate < '2023-03-03' order by trackdate desc limit
1
```

사용한 sql문은 다음과 같다. tracking entity를 trackdate로 정렬한 뒤 trackdate가 2023년 3월 3일보다 작은, 즉 2023년 3월 3일 이전에 마지막으로 truck_1721이 배송한 건을 찾는다. 찾은 배송 건에 대하여 shipment_ID와 trackdate를 추출하여 출력한다. 결과는 다음과 같다.

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

        1. TYPE I
        2. TYPE II
        3. TYPE III
        4. TYPE IV
        5. TYPE V
        0. QUIT
Select query type : 1
----- Subtypes in TYPE I -----
        1. TYPE I-1.
        2. TYPE I-2.
        3. TYPE I-3.
Select query type of TYPE I : 3
---- TYPE I-3 ----

**Find the last successful delivery by that truck prior to the crash**
Shipment_ID of last successful delivery process : S24680 at 2023-02-27
```

배송 ID S24680이 2023년 2월 27일에 truck_1721에 의해 마지막으로 배송되었다.

2) TYPE 2 : Find the customer who has shipped the most packages in the past year

```
select customer_ID, name, cummulative_shipment
from customer
where cummulative_shipment = (select max(cummulative_shipment) from customer)
```

사용한 sql문은 다음과 같다. customer entity에서 cummulative_shipment가 최대값인 tuple의 customer_ID, name, cummulative를 추출하였다. 또한, cummulative_shipment가 최대값을 가지는 고객이 여러명이 있는 경우가 있을 수 있으므로 루프를 돌며 해당되는 모든 경우를 출력하도록 했다. 결과는 다음과 같다.

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

        1. TYPE I
        2. TYPE II
        3. TYPE III
        4. TYPE IV
        5. TYPE V
        0. QUIT
Select query type : 2
---- TYPE II ----

**Find the customer who has shipped the most packages in the past year**
| A1234567 chulsoo kim ; cummulative_shipment : 4 |
```

해당 고객은 누적 배송건수가 4건으로, 가장 많은 누적 배송건수를 가지고 있는 고객이다.

3) TYPE 3 : Find the customer who has spent the most money on shipping the past year

```
select customer_ID, name, cummulative_cost
from customer
where cummulative_cost = (select max(cummulative_cost) from customer)
```

사용한 sql문은 다음과 같다. query type II와 유사하게, 누적금액의 최대값과 같은 누적금액을 가지는 고객의 정보를 추출하였다. 결과는 다음과 같다.

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

        1. TYPE I
        2. TYPE II
        3. TYPE III
        4. TYPE IV
        5. TYPE V
        0. QUIT
Select query type : 3
---- TYPE III ----

**Find the customer who has spent the most money on shipping in the past year**
| A2345678 jeongsoo choi ; cummulative_cost : 51000 |
```

해당 고객은 누적 주문금액이 51000원으로, 가장 많은 누적 주문금액을 가지고 있는 고객이다.

4) TYPE IV : Find the packages that were not delivered within the promised time

```
select distinct P.package_ID, P.customer_ID
from package P natural join tracking T natural join shipment S
where P.package_ID = T.package_ID and T.shipment_ID = S.shipment_ID and T.shipping_key = 'complete' and T.trackdate > S.prearranged_time
```

사용한 sql문은 다음과 같다. package, tracking, shipment entity를 natural join한 뒤 package의 package_ID와 tracking의 package_ID, tracking의 shipment_ID와 shipment의 shipment_ID가 같으며 shipping_key가 complete이고 complete된 trackdate가 shipment의 prearranged_time을 초과한 배송들의 package_ID와 customer_ID를 추출하였다. 결과는 다음과 같다.

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

      1. TYPE I
      2. TYPE II
      3. TYPE III
      4. TYPE IV
      5. TYPE V
      0. QUIT
Select query type : 4
---- TYPE IV ----

**Find the packages that were not delivered withen the promised time**
| package_ID : P913748745, customer_ID : A1234567 | package_ID : P954883923, customer_ID : A6789012 | package_ID : P976584293, customer_ID : A2345678 |
```

해당 배송건과 배송의 고객은 배송예정시간을 초과하여 배송된 경우에 해당한다.

5) TYPE V : Generate the bill for each customer for the past month

먼저, 현재 일자가 2023년 4월이라는 가정 아래 2023년 3월의 bill을 생성하도록 했다. query type 5를 처음 실행했을 때 고객의 ID와 bill의 type을 먼저 입력받고 제목을 생성했다 코드는 다음과 같다.

```
printf("---- TYPE V ----\n\n");
printf("**Generate the bill for each customer for the past month**\n");
printf(" Which Customer? (Customer ID) : ");
string customer_ID;
int bill_type;
cin >> customer_ID;
printf(" Which type? (type 1: simple, type 2: by type of service, type 3: by each shipment) : ");
scanf("%d", &bill_type);
string st_date = "2023-03-01";
string ed_date = "2023-03-31";
// title
string query = "select name from customer where customer_ID = '" + customer_ID + "'";
string name;
int state = 0;
state = mysql_query(connection, query.c_str());
if (state == 0)
{
    int flag = 0;
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        if (!flag) flag++;
        name = string(sql_row[0]);
    }
    mysql_free_result(sql_result);
    if (!flag) {
        printf("There is no customer who has customer ID : %s\n\n", customer_ID.c_str());
        continue;
    }
}
printf("\n\t\t%s %s bill\n\n", customer_ID.c_str(), name.c_str());
```

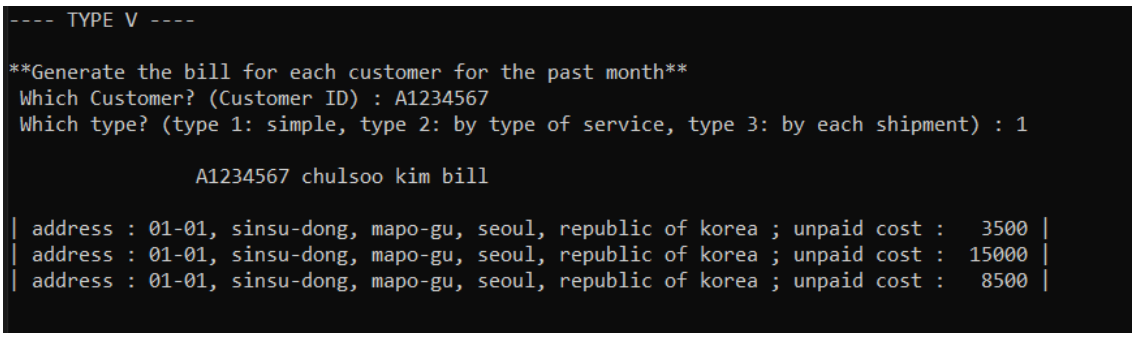
customer_ID와 bill_type을 입력받고, select name from customer where customer_ID = (입력받은 customer_ID)의 query를 통해 고객의 이름을 저장했다. 만약 query의 결과가 없다면 "there is no customer who has customer ID : ..."를 출력하고 query type 선택 화면으로 돌아가도록 했다.

(5-1) simple bill

```
select distinct address, unpaid_cost
from payment
where customer_ID = 'input costumer_ID' and payment_date >= '2023-03-01' and payment_date <= '2023-03-31'
```

simple bill에서 사용한 쿼리문은 다음과 같다. payment entity에서 costumer_ID가 입력받은 ID이고 2023년 3월에 해당되는 주문의 address와 unpaid_cost를 추출하여 알맞게 출력했다.

결과는 다음과 같다.



주문한 주소가 다를 경우를 대비하여, 각각의 주문에 대한 주소와 미지불 금액을 따로 출력하도록 하였다.

(5-2) A bill listing charges by type of service

A bill listing charges by type of service에서는 payment ID, total cost, payment type, payment date, unpaid cost, package ID, shipper_ID, shipper name, account number를 각각 query로 추출하여 알맞게 출력하였다. 서비스는 어떤 배송업체를 이용했는지를 나타내었다. 또한 payment ID, total cost, payment type, unpaid cost, package ID는 bill에 들어가야 할 필수 요소라고 판단하였고 해당 type의 bill에 shipper_ID, shipper name, account number가 들어가는 것이 적절하다고 판단했다.

(5-2-1) query for payment_ID

```
select payment_ID
from payment
where customer_ID = 'input customer_ID' and payment_date >= '2023-03-01' and payment_date
<= '2023-03-31' order by payment_ID
```

사용한 sql문은 다음과 같다. 입력받은 customer_ID와 같은 customer_ID 및 2023년 3월에 해당하는 payment_ID를 추출하였다. 코드는 다음과 같다.

```
// payment id
query.clear();
query = "select payment_ID from payment where customer_ID = '" + customer_ID + "' and payment_
vector<string> pay_ID;
state = 0;
state = mysql_query(connection, query.c_str());
if (state == 0)
{
    int flag = 0;
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        if (!flag) flag++;
        string tmp(sql_row[0]);
        pay_ID.push_back(tmp);
    }
    mysql_free_result(sql_result);
    if (!flag) {
        printf("There is no payment for this customer.\n\n", customer_ID.c_str());
        continue;
    }
}
printf("|");
for (int i = 0; i < pay_ID.size(); i++) {
    printf(" payment ID : %12s \t |", pay_ID[i].c_str());
}
```

코드에서 pay_ID vector를 생성하여 query의 결과를 push_back하고 순차적으로 출력하였다. vector에 저장한 pay_ID는 추후의 query에서 사용된다.

(5-2-2) query for total_cost

```
select total_cost
from payment
where customer_ID = 'input customer_ID' and payment_date >= '2023-03-01' and payment_date
<= '2023-03-31' order by payment_ID
```

payment_ID와 비슷한 방법으로 sql문을 작성하였다. 입력받은 customer_ID와 같은 customer_ID 및 2023년 3월에 해당하는 total_cost를 추출하여 출력하였다.

(5-2-3) query for payment type

```
select payment_type
from payment
where customer_ID = 'input customer_ID' and payment_date >= '2023-03-01' and payment_date
<= '2023-03-31' order by payment_ID
```

payment_ID와 비슷한 방법으로 sql문을 작성하였다. 입력받은 customer_ID와 같은 customer_ID 및 2023년 3월에 해당하는 payment type을 추출하여 출력하였다.

(5-2-4) query for payment date

```
select payment_date

from payment

where customer_ID = 'input customer_ID' and payment_date >= '2023-03-01' and payment_date
<= '2023-03-31' order by payment_ID
```

payment_ID와 비슷한 방법으로 sql문을 작성하였다. 입력받은 customer_ID와 같은 customer_ID 및 2023년 3월에 해당하는 payment_date를 추출하여 출력하였다.

(5-2-5) query for unpaid cost

```
select unpaid_cost

from payment

where customer_ID = 'input customer_ID' and payment_date >= '2023-03-01' and payment_date
<= '2023-03-31' order by payment_ID
```

payment_ID와 비슷한 방법으로 sql문을 작성하였다. 입력받은 customer_ID와 같은 customer_ID 및 2023년 3월에 해당하는 unpaid_cost를 추출하여 출력하였다.

(5-2-6) query for package_id

```
select package_ID

from payment

where customer_ID = 'input customer_ID' and payment_date >= '2023-03-01' and payment_date
<= '2023-03-31' order by payment_ID
```

payment_ID와 비슷한 방법으로 sql문을 작성하였다. 입력받은 customer_ID와 같은 customer_ID 및 2023년 3월에 해당하는 package_ID를 추출하여 출력하였다.

(5-2-6) query for shipper ID

```
select shipper_ID
from payment_through_shipper
where payment_ID = 'pay_ID[i]'
```

사용한 sql문은 다음과 같다. pay_ID vector의 size만큼 루프를 돌며 payment_through_shipper entity에서 payment_ID가 pay_ID의 i번째 원소와 같으면 알맞게 출력하였다. 만약 결과가 없을 경우 "did not pay by shipper"를 출력하였다. 코드는 다음과 같다.

```
printf("|");
vector<string> shipper;
int flag = 0;
for (int i = 0; i < pay_ID.size(); i++) {
    query.clear();
    query = "select shipper_ID from payment_through_shipper where payment_ID = '" + pay_ID[i] + "'";
    string sid;
    state = 0;
    state = mysql_query(connection, query.c_str());
    if (state == 0)
    {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            if (!flag) flag++;
            string tmp(sql_row[0]);
            sid = tmp;
            shipper.push_back(tmp);
        }
        mysql_free_result(sql_result);
        if (!flag) shipper.push_back("none");
    }
    if (flag) printf(" shipper_ID : %12s\t |", sid.c_str());
    else printf(" did not pay by shipper    \t |");
}
printf("\n");
```

코드에서 query의 결과를 shipper vector에 push_back하여 저장하였다. shipper vector는 바로 뒤의 shipper_name을 추출하는 query에서 사용하였다. 만약 query의 결과가 없을 경우 "none"을 삽입하였다.

(5-2-7) query for shipper name

```
select name
from shipper
where shipper_ID = 'shipper[i]'
```

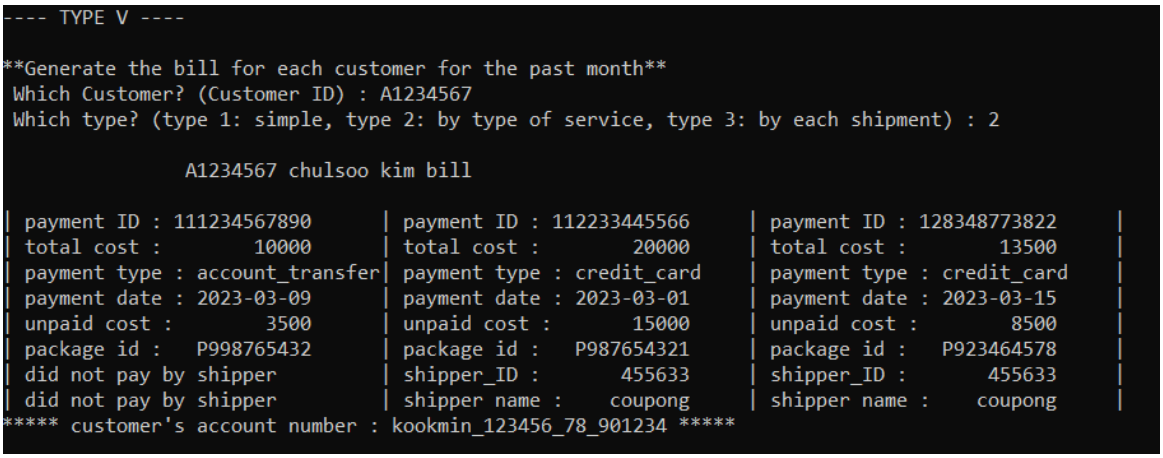
사용한 sql문은 다음과 같다. shipper vector의 size만큼 루프를 돌며 shipper entity의 shipper_ID가 shipper vector의 i번째 원소와 같을 경우 결과를 출력하고, 해당 원소가 "none"일 경우 "did not pay by shipper"를 출력하였다.

(5-2-8) query for account number

```
select account
from customer
where customer_ID = 'input customer_ID'
```

사용한 sql문은 다음과 같다. 입력받은 customer_ID와 같은 customer_ID를 가진 account를 추출하여 출력하였다.

최종적으로, A bill listing charges by type of service를 출력한 결과는 다음과 같다.



해당 bill은 customer_ID가 A1234567인 chulsoo kim의 bill listing charges by type of service이다.

(5-3) An itemize billing listing each individual shipment and the charges for it

An itemize billing listing each individual shipment and the charges에서는 payment ID, total cost, payment type, payment date, unpaid cost, package ID, shipment_ID, content, value를 각각 query로 추출하여 알맞게 출력하였다. 이 중 payment ID부터 package ID는 A bill listing

charges by type of service에서의 query와 동일하다. 다만, 뒤의 query에서 사용할 정보인 package_ID를 pack_id vector에 저장하였다. 따라서 shipment_ID, content, value를 추출한 query만을 설명하였다.

(5-3-1) query for shipment ID

```
select distinct shipment_ID
from tracking
where package_ID = 'pack_id[i]'
```

사용한 sql문은 다음과 같다. pack_id vector의 size만큼 루프를 돌며 tracking entity에서 package_ID가 pack_id의 i번째 원소와 같은 shipment_ID를 unique하게 추출하였다. 코드는 다음과 같다.

```
// shipment_ID
printf("|");
for (int i = 0; i < pack_id.size(); i++) {
    query.clear();
    query = "select distinct shipment_ID from tracking where package_ID = '" + pack_id[i] + "'";
    state = 0;
    state = mysql_query(connection, query.c_str());
    if (state == 0)
    {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            printf(" shipment id : %11s \t |", sql_row[0]);
        }
        mysql_free_result(sql_result);
    }
}
printf("\n");
```

(5-3-2) query for content

```
select content
from package
where package_ID = 'pack_id[i]'
```

사용한 sql문은 다음과 같다. 루프를 돌며 package entity에서 package_ID가 pack_id의 i번째 원소와 같은 content를 추출하여 출력하였다.

(5-3-3) query for value

```
select value
from package
where package_ID = 'pack_id[i]'
```

content를 추출하는 query와 비슷하게 sql문을 사용하였다. 루프를 돌며 package entity에서 package_ID가 pack_id의 i번째 원소와 같은 value를 추출하여 출력하였다.

최종적으로, An itemize billing listing each individual shipment and the charges를 출력한 결과는 다음과 같다.

```
---- TYPE V ----

**Generate the bill for each customer for the past month**
 Which Customer? (Customer ID) : A1234567
 Which type? (type 1: simple, type 2: by type of service, type 3: by each shipment) : 3

               A1234567 chulsoo kim bill

| payment ID : 111234567890      | payment ID : 112233445566     | payment ID : 128348773822     |
| total cost :        10000      | total cost :        20000     | total cost :        13500     |
| payment type : account_transfer| payment type : credit_card    | payment type : credit_card    |
| payment date : 2023-03-09      | payment date : 2023-03-01     | payment date : 2023-03-15     |
| unpaid cost :        3500      | unpaid cost :       15000     | unpaid cost :        8500     |
| package id :   P998765432      | package id :   P987654321     | package id :   P923464578     |
| shipment id :      S12345      | shipment id :      S13579     | shipment id :      S48372     |
| content :      wine_glass      | content :            book     | content :        SSD_card     |
| value :             10000      | value :             35000     | value :            200000     |
```

해당 bill은 customer_ID가 A1234567인 chulsoo kim의 itemize billing listing each individual shipment and the charges이다.