CSE4110 - Database System

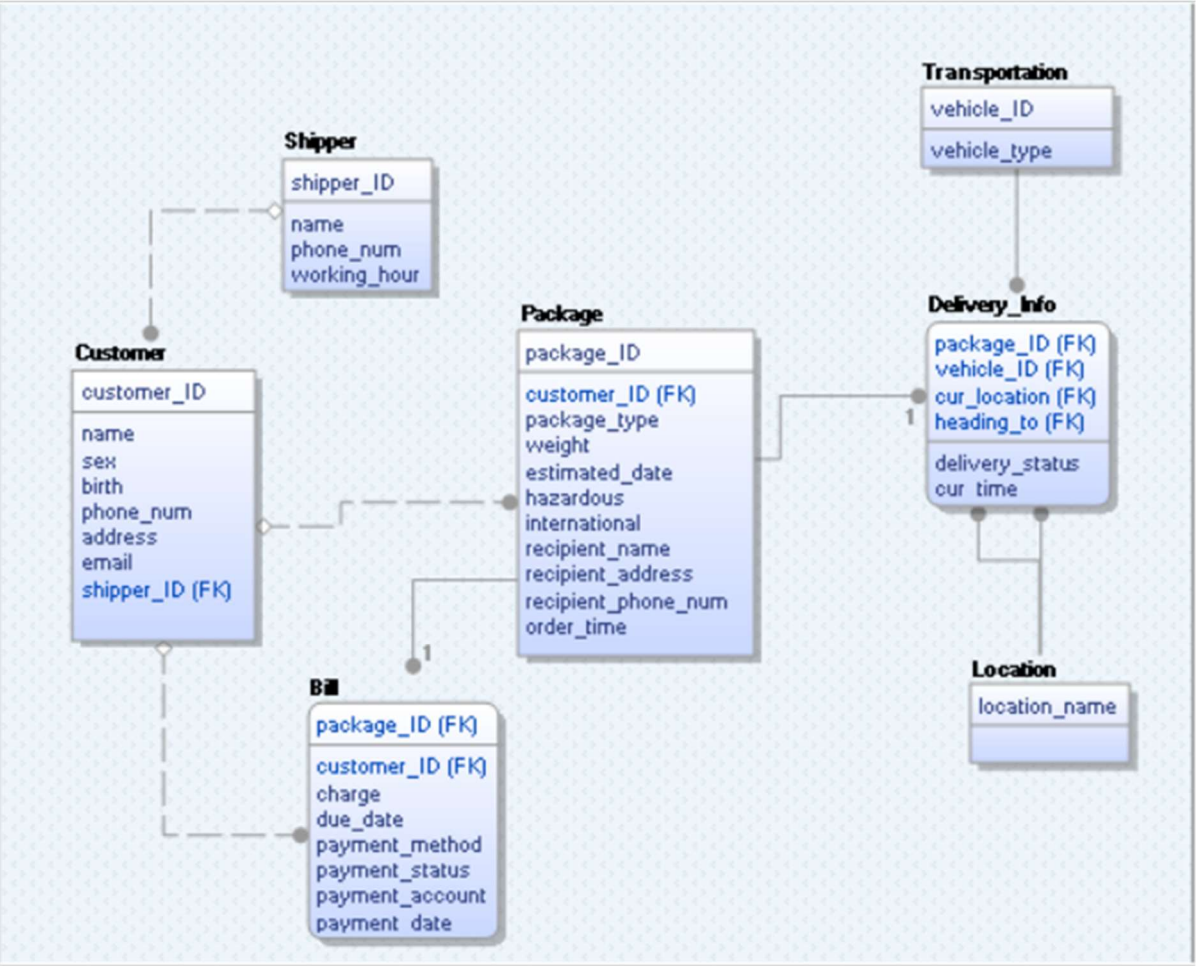
Project2. Normalization And Query Processing

20180032

남기동

목차

1.1 Before BCNF Decomposition



위의 표는 Project1에서 설계한 ER Model을 토대로 Erwin을 통해 만든 Logical Schema이다. 해당 Logical Schema를 Physical Schema로 전환하고 Database에 연결하여 query를 실행하기 이전에 필요한 단계는 모든 Entity가 BCNF인지 체크하고 그렇지 않은 Entity에 대해서 Decomposition을 수행하는 것이다.

1.2 BCNF Testing

1) Customer

Customer Entity는 기본적으로 고객의 정보를 나타내는 Entity이다. 속성으로는 customer_ID, name, sex, birth, phone_num, address, email, shipper_ID를 갖고 있다. Primary key는 customer_ID이며 Foreign key로 Shipper의 primary key인 shipper_ID를 갖고 있다.

Customer Entity에 존재하는 Functional dependency는 (customer_ID → name, sex, birth, phone_num, address, email, shipper_ID) 뿐이다. 해당 함수 종속성은 customer_ID가 primary key이

기 때문에 BCNF의 조건 중 하나인 superkey를 만족하기 때문에 Customer Entity는 BCNF인 것을 확인할 수 있다.

2) Shipper

Shipper Entity는 배달 기사의 정보를 저장한다. 속성으로는 shipper_ID, name, phone_num, work_hour을 갖고 있으며 Primary key는 shipper_ID이다.

Shipper Entity에 존재하는 Functional Dependency는 (shipper_ID → name, phone_num, work_hour) 뿐이다. 해당 함수 종속성은 shipper_ID가 primary key이기 때문에 BCNF의 조건 중 하나인 superkey를 만족하기 때문에 Shipper Entity도 BCNF인 것을 확인할 수 있다.

3) Package

Package Entity는 이번 DB 프로젝트에서 핵심이 되는 entity로 패키지의 정보를 저장한다. 속성으로는 Package_ID, Package_type, Weight, Estimated_date, Hazardous, International, recipient_name, recipient_address, recipient_phone_num, order_time, customer_ID를 갖고 있다. Primary key는 Package_ID이며 foreign key로 customer_ID를 customer entity에서 참조하고 있다.

Package Entity에 존재하는 Functional Dependency는 (package_ID → Package_type, Weight, Estimated_date, Hazardous, International, recipient_name, recipient_address, recipient_phone_num, order_time, customer_ID) 뿐이다. 해당 함수 종속성은 package_ID가 primary key이기 때문에 BCNF의 조건 중 하나인 superkey를 만족하기 때문에 Package Entity도 BCNF인 것을 확인할 수 있다.

Package의 속성들 중 recipient, 즉 수령인과 관련된 속성들 사이에 Functional Dependency가 있지 않는가 하는 생각을 할 수 있다. 하지만 recipient_name만으로는 수령인의 주소와 전화번호를 확정지을 수 없다. 일상에서 사용하는 배달 서비스를 생각해봤을 때, 수령인의 이름이 중복될 수도 있으며 대리 수령과 같은 맥락에서는 다른 주소를 사용하거나 다른 전화번호를 기입하기도 하기 때문이다. 따라서 Package Entity는 여전히 BCNF를 만족한다는 결론을 내렸다.

4) Bill

Bill은 고객이 주문한 package에 대해 청구서의 정보를 저장하는 용도로 사용하는 Entity이다. 속성으로는 package_ID, customer_ID, charge, due_date, payment_method, payment_status, payment_account, payment date를 갖고 있다. Primary key는 Package에서 참조한 foreign key이기도 한 package_ID이며 customer_ID도 customer에서 참조한 foreign key이다.

Bill Entity에 존재하는 Functional Dependency는 (package_ID → customer_ID, charge, due_date,

payment_method, payment_status, payment_account, payment date) 뿐이다. 해당 함수 종속성은 package_ID가 primary key이기 때문에 BCNF의 조건 중 하나인 superkey를 만족하기 때문에 Bill Entity도 BCNF인 것을 확인할 수 있다.

Payment와 관련하여 method, status, account, date가 있기 때문에 이와 관련된 FD가 있을까 생각해보았지만 지불 방법, 지불 상태, 지불 계좌, 지불 날짜에서는 어떤 것이 다른 것들을 결정하는 관계를 생각할 수 없다. 지불 계좌에 따라서도 여러 bill이 있을 수 있으며 지불 방법과 지불 날짜 지불 상태는 당연히 중복될 수 있기 때문이다. 따라서 기존에 설명한 FD이외의 FD는 없으므로 Bill Entity는 여전히 BCNF를 만족한다는 결론을 내렸다.

5) Delivery_Info

Delivery_Info Entity는 package가 어떤 경로로 운송되고 있는지, 그리고 현재 어떤 상태에 있는지를 추적하기 위해 만들어진 entity이다. 정보의 중복이 일부 발생할 수 있지만 명세서에서 배달 업체는 배송되는 과정의 세부 사항들을 추적하고 있어야 한다고 했기 때문에 따로 entity로 만들었다. Delivery_Info는 속성으로 package_ID, vehicle_ID, cur_location, heading_to, delivery_status, cur_time을 갖는다. Primary key로는 foreign key이기도 한 package_ID, vehicle_ID, cur_location, heading_to을 갖는다. 굳이 네 가지 속성을 primary key로 갖는 이유는 delivery info에 정보를 저장할 때 package_Id 만으로 식별할 수 없기 때문이다. 명세서에서 배송 과정에 대한 세부정보를 가져야 한다고 했기 때문에 delivery info에 저장되는 정보는 package 별로 1개씩만 있지 않다. 즉, 같은 차량이 같은 패키지를 담고 장소를 이동하기 때문에 package_id 만으로는 delivery_info에 저장되는 튜플들이 유일하게 식별되지 않는다. 같은 지역을 두 번 지나가는 경우도 있기 때문에 cur_location에 추가로 heading_to도 추가하여 총 네 가지의 속성(package_id, cur_location, heading_to, vehicle_id)의 조합으로 delivery_info의 튜플을 유일하게 식별할 수 있다.

Delivery_Info Entity에 존재하는 Functional Dependency는 (package_id, cur_location, heading_to, vehicle_id → delivery_status, cur_time)이다. 해당 함수 종속성은 4개의 속성이 primary key이기 때문에 BCNF의 조건 중 하나인 superkey를 만족하기 때문에 Delivery_Info Entity도 BCNF인 것을 확인할 수 있다.

6) Transportation

Transportation Entity는 Delivery_Info에서 package가 배달되는 경우 어떤 운송 수단을 이용하고 있는지 정보를 나타내고자 할 때 참조하기 위해 만들어진 entity이다. 속성으로는 vehicle_ID, vehicle_type을 갖는다. Vehicle_ID가 primary key 역할을 한다.

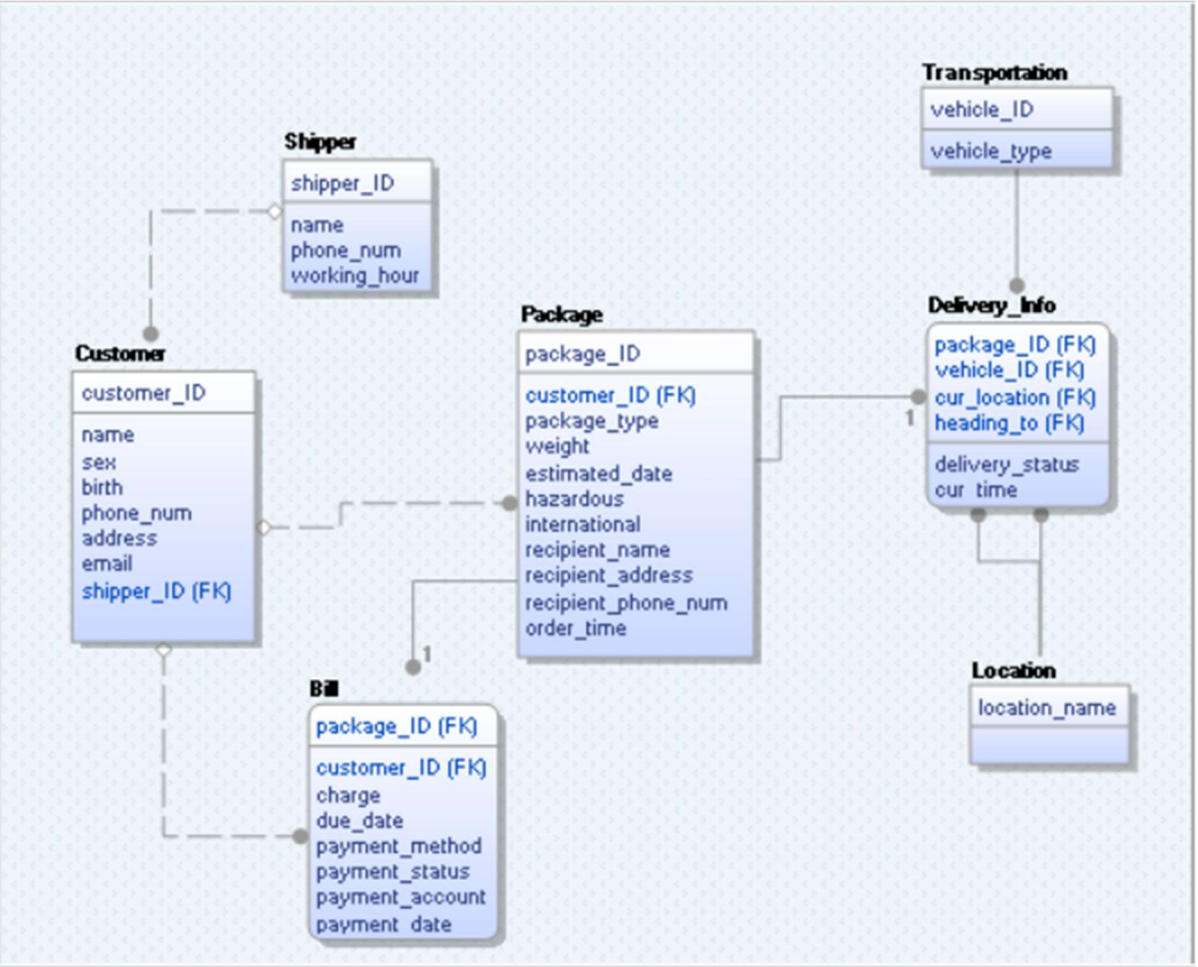
Transportation Entity에 존재하는 Functional Dependency는 (vehicle_ID → vehicle_type)뿐이다. 해

당 함수 종속성은 vehicle_ID가 primary key이기 때문에 BCNF의 조건 중 하나인 superkey를 만족하기 때문에 Transportation Entity도 BCNF인 것을 확인할 수 있다.

7) Location

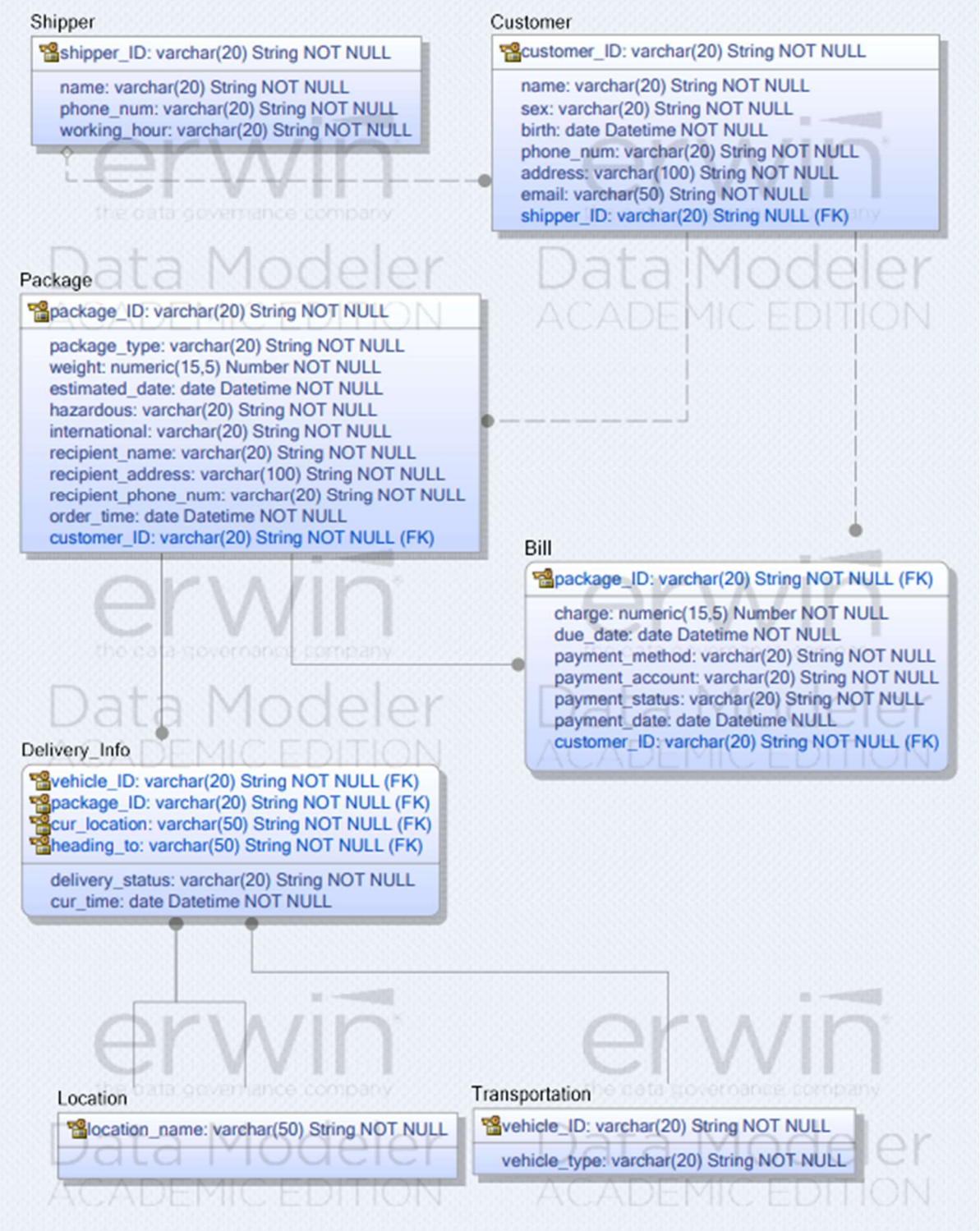
Location Entity는 Delivery_Info에서 현재 package가 어디 위치해 있는지를 나타내고자 할 때 참조하기 위해 만든 엔티티이다. 너무 자세한 주소를 저장하기 보다는 특정 도시에 있다는 정보만 저장하면 되기 때문에 많은 tuple이 저장되지는 않을 것으로 예상된다. 속성으로는 location_name만을 갖는다. 속성이 하나 밖에 없으니 따로 함수 종속성이 존재하지 않고 따라서 BCNF로 볼 수 있다.

1.3 After BCNF Decompositon



BCNF Testing을 통해 7개의 Entity를 검사하였지만 모두 이미 BCNF를 만족하기 때문이 1.1에서 사용한 Relational Logical Schema가 이후 Physical Schema를 만드는데 사용될 것이다.

2.1 Physical Schema Diagram



위의 그림은 1.3에서 사용하기로 정한 Relational Logical Schema를 사용하여 Physical Schema를 설계하여 만든 Diagram이다. 7개의 엔티티에 대하여 각 속성의 type과 domain, NULL여부, 제약조건, relationship type에 대해서는 아래에서 차례대로 설명할 것이다.

2.2 Data Type, Domain, Constraints, Null Allowing

1) Customer

```
create table Customer(
        customer_ID  varchar(20)  NOT NULL ,
        name  varchar(20)  NOT NULL ,
        sex  varchar(20)  NOT NULL ,
                check (sex in ('male', 'female')),
        birth  date  NOT NULL ,
        phone_num  varchar(20)  NOT NULL ,
        address  varchar(100)  NOT NULL ,
        email  varchar(50)  NOT NULL ,
        shipper_ID  varchar(20)  NULL ,
        primary key(customer_ID),
        foreign key(shipper_ID) references Shipper(shipper_ID)
);
```

우선 기본적으로 대부분의 속성들은 String을 Domain으로 갖고 있으며 varchar(20)을 데이터 타입으로 갖고 NULL을 허용하지 않는다. NULL을 대부분 허용하지 않는 이유는 고객의 기본 정보들을 알아야 고객을 관리하고 배송에 문제가 되지 않도록 정보를 이용해야 하기 때문이다.

Customer_ID는 Primary key이기 때문에 우선 NULL이 허용되지 않으며 ID를 string으로 20글자 이내로 받는다. Name도 동일하고 sex도 동일하다. 그러나 sex의 경우 male이랑 female 2개의 옵션만 있기 때문에 check를 사용하여 제약조건을 걸어 놓았다. Birth는 고객의 생년월일을 저장하기 때문에 Datetime Domain의 date를 데이터 타입으로 갖으며 기본 정보이기 때문에 이 역시도 NULL을 허용하지 않는다. 주소와 이메일은 데이터의 길이가 더 길어질 수 있으니 100글자, 50글자로 제한을 늘렸다. Shipper_ID도 customer_ID와 유사하게 20개를 갖는데 shipper이 배정되지 않는 경우도 있기 때문에 NULL을 허용했다.

2) Shipper

```
create table Shipper(
        shipper_ID  varchar(20)  NOT NULL ,
        name  varchar(20)  NOT NULL ,
        phone_num  varchar(20)  NOT NULL ,
        working_hour  varchar(20)  NOT NULL ,
                check (working_hour in ('day', 'night', 'overnight')),
        primary key(shipper_ID),
        foreign key
);
```

Shipper의 속성은 모두 string domain에 속하며 varchar(20)을 데이터 타입으로 갖는다. Shipper_ID는 primary key이므로 NULL을 허용하지 않으며 이름, 전화번호, 근무 시간도 기본 정보이고 배송과 관련하여 활용될 수 있는 정보이기 때문에 NULL을 허용하지 않는 것으로 설정하였다. 단, working hour는 낮 근무, 저녁 근무, 새벽 근무만 있기 때문에 check를 넣어 제약조건을 달았다.

3) Package

```
create table Package(
        package_ID  varchar(20)  NOT NULL ,
        package_type  varchar(20)  NOT NULL ,
                check (package_type in ('envelope', 'small box', 'large box')),
        weight  numeric(15,5)  NOT NULL ,
        estimated_date  date  NOT NULL ,
        hazardous  varchar(20)  NOT NULL ,
        international  varchar(20)  NOT NULL ,
        recipient_name  varchar(20)  NOT NULL ,
        recipient_address  varchar(100)  NOT NULL ,
        recipient_phone_num  varchar(20)  NOT NULL ,
        order_time  date  NOT NULL ,
        customer_ID  varchar(20)  NOT NULL ,
        primary key (package_ID)
        foreign key (customer_ID) references Customer(customer_ID)
);
```

Package도 대부분 기본 설정인 string을 domain으로 하며 varchar(20)을 데이터타입으로 하고 배송에 중요한 정보들을 담기 때문에 모두 NULL을 허용하지 않는다. 살펴볼 만한 것으로는 package_type이 envelope, small box, large box 이렇게 세 종류가 있기 때문에 이에 해당하는 것만 허용하도록 제약조건을 달았고 weight는 Number Domain의 numeric(15,5)를 사용하여 숫자를 값으로 갖도록 하였다. Estimated_date는 도착 예정 시간으로 Datetime Domain의 date를 사용하고 있다. Order_time도 주문 날짜를 반영하므로 Datetime Domain의 date를 데이터 타입으로 사용한다.

4) Bill

```
create table Bill(
        charge  numeric(15,5)  NOT NULL ,
        due_date  date  NOT NULL ,
        payment_method  varchar(20)  NOT NULL ,
                check (payment_method in ('cash', 'credit card', 'transfer')),
        payment_account  varchar(20)  NOT NULL ,
        payment_status  varchar(20)  NOT NULL ,
                check (payment_status in ('not yet', 'prepaid', 'montly')),
        payment_date  date NULL ,
        package_ID  varchar(20)  NOT NULL ,
        customer_ID  varchar(20)  NOT NULL ,
        primary key (package_ID),
        foreign key (package_ID) references Package(package_ID),
        foreign key (customer_ID) references Customer(customer_ID)
);
```

Charge는 청구 비용을 나타내므로 Number Domain의 numeric(15,5)을 사용하였고 due_date는 비용 지불 기한이므로 Datetime Domain의 date를 사용했다. Payment_method는 어떤 방식으로 지

불할 것인지를 나타내기 때문에 cash, credit card, trasfer로 제약을 두었고 payment_status는 지불 상태를 나타내기 때문에 아직 지불되지 않았다는 not yet, 이미 지불되었다는 prepaid, 매달 결제된다는 montly에 해당하도록 제약 조건을 걸어두었다. Payment_date 지불 날짜이므로 Datetime Domain의 Date를 데이터 타입으로 갖는데 NULL을 허용하였다. 그 이유는 payment_status가 not yet으로 아직 결제되지 않은 상태라면 결제 날짜가 없을 수 있기 때문이다.

5) Delivery_Info

```
create table Delivery_Info(
        delivery_status  varchar(20)  NOT NULL ,
                check (delivery_status in ('store', 'move', 'delay', 'crash', 'done')),
        cur_time  date  NOT NULL ,
        vehicle_ID  varchar(20)  NOT NULL ,
        cur_location  varchar(50)  NOT NULL ,
        package_ID  varchar(20)  NOT NULL ,
        heading_to  varchar(50)  NOT NULL ,
        primary key (vehicle_ID, package_ID, cur_location, heading_to),
        foreign key (vehicle_ID) references Transportation(vehicle_ID),
        foreign key (cur_location) references Location(location_name),
        foreign key (package_ID) references Package(package_ID),
        foreign key (heading_to) references Location(location_name)
);
```

Delivery_status는 string domain의 varchar(20)을 데이터 타입으로 하는데 store, move, delay, crash, done이라는 상태만 가질 수 있도록 제약을 걸었다. Cur_time은 Datetime의 date를 데이터타입으로 갖으며 location에 해당하는 cur_location과 heading_to는 location의 이름이 길어질 수도 있으니 여유롭게 50글자로 설정하였다. 나머지는 모두 string domain의 varchar(20)이며 NULL을 허용하지 않는 것으로 설정하였다.

6) Transportation

```
create table Transportation(
        vehicle_ID  varchar(20)  NOT NULL ,
        vehicle_type  varchar(20)  NOT NULL ,
                check (vehicle_type in ('truck', 'plane', 'train')),
        primary key (vehicle_ID)
);
```

Transportation의 속성으로는 vehicle_ID와 type 두 개가 있다. 둘 다 string domain의 varchar(20)을 데이터 타입으로 갖는다. 또한 vehicle_type의 경우 truck, plane, train 이렇게 세 종류에만 해당하도록 제약을 걸었다.

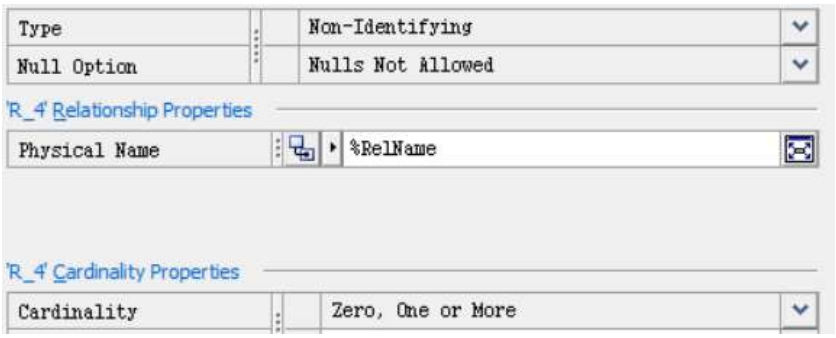
7) Location

```
create table Location(
        location_name  varchar(50)  NOT NULL ,
        primary key  ([location_name)
);
```

마지막으로 location은 location_name만을 속성으로 갖는데 앞서 이야기했듯 지명이 길 수 있기 때문에 varchar(50)으로 여유를 두었고 primary key이기 때문에 NULL은 허용되지 않는다.

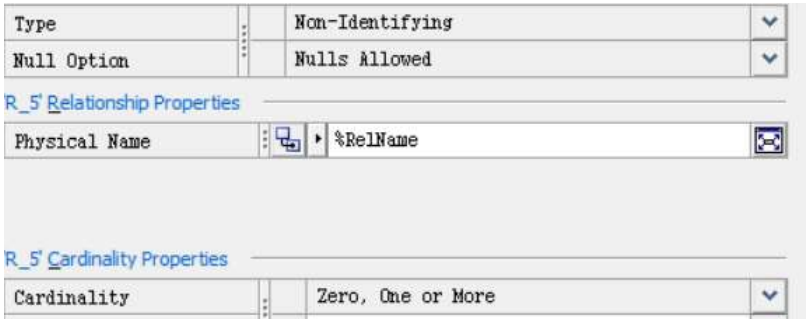
2.3 Relationship Type

1) Customer - Package



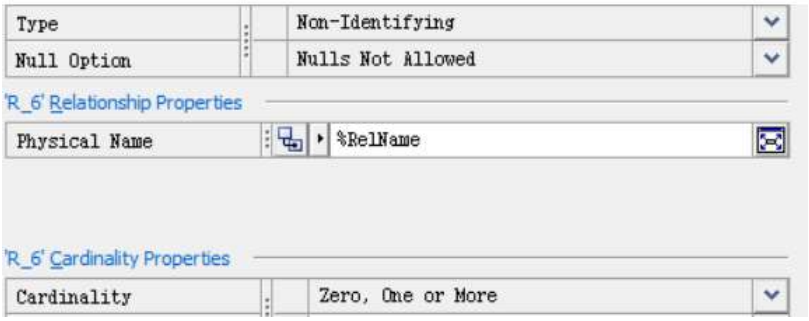
우선 customer-package의 relationship type은 many to one이다. Cardinality를 살펴보면, customer는 package를 0~many까지 가질 수 있고 package는 customer를 무조건 1명 가져야 하기 때문에 one-to zero, one or more이 된다. 그리고 total이기 때문에 Nulls Not Allowed로 설정했다.

2) Customer - Shipper



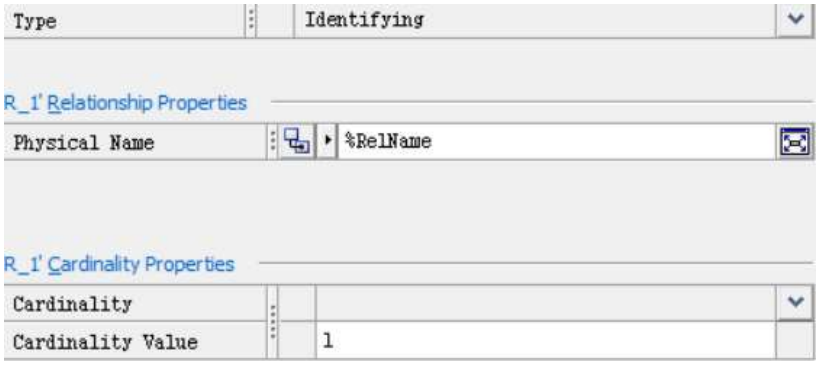
shipper – customer는 one-to-many의 관계이며 shipper는 customer를 0~many를 가질 수 있으며 customer는 0~1의 shipper를 가질 수 있다. 따라서 zero, one or more이 들어가야 한다. 그리고 total이 아니기 때문에 Nulls allowed로 설정했다.

3) Customer – Bill



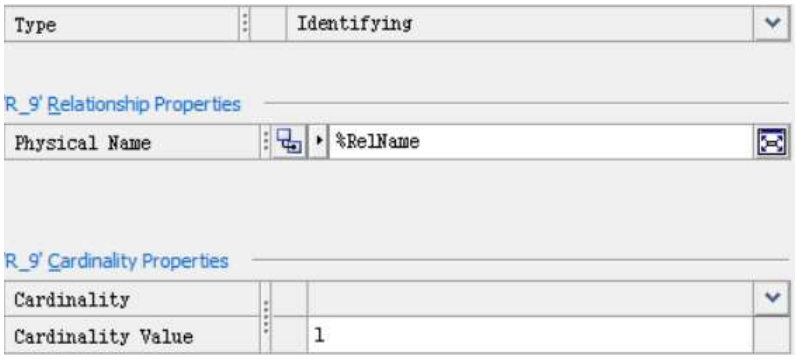
customer는 bill을 0~many개 가질 수 있지만 bill은 1개의 고객만을 가질 수 있다. 따라서 one-to-zero, one or more을 선택한 것을 확인할 수 있다. 그리고 total이기 때문에 Nulls Not Allowed로 설정했다.

4) Package – Bill



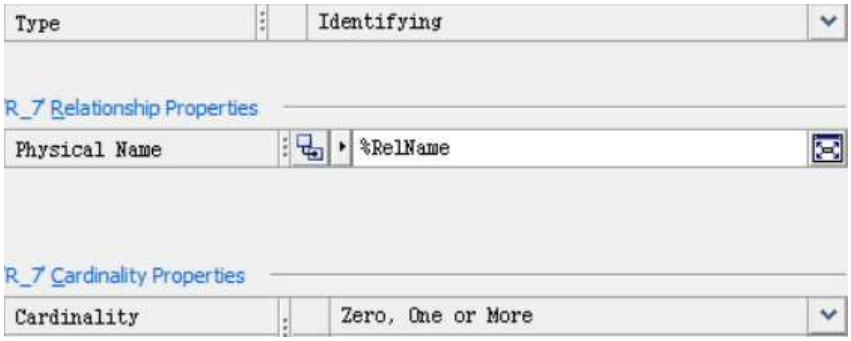
Package – bill은 one-to-one의 관계이다. 따라서 Cardinality에 1을 기입하여 이를 표현하였다. 또한 Bill은 package의 primary key로 정의되는 Weak Entity이기 때문에 'Identifying'으로 표시되도록 설정하였다.

5) Package – Delivery_Info



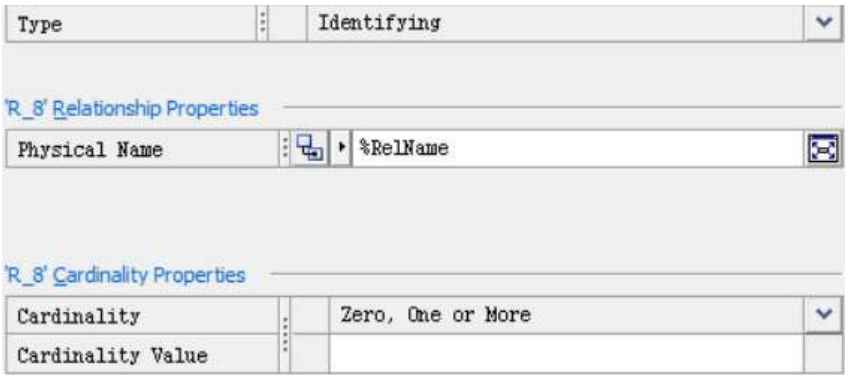
package-delivery_info는 one-to-one의 관계를 가졌기 때문에 cardinality value에 1을 설정하였다. 또한 Delivery_Info는 package의 primary key로 정의되는 Weak Entity이기 때문에 'Identifying'으로 표시되도록 설정하였다.

6) Delivery_Info – Transportation



Transportation 1개가 Delivery_Info의 튜플에 하나도 적용되지 않거나 하나 혹은 여러 개에 적용될 수 있기 때문에 one-to-zero, one or more로 설정되었다. 또한 Delivery_Info는 Transportation의 primary key에 의존하기 때문에 Weak entity이다. 따라서 vehicle_ID가 foreign key로 Delivery_Info의 primary key를 구성하고 있으며 identifying으로 타입이 설정되어 있다.

7) Delivery_Info – Location



Delivery_Info는 Location의 primary key(location_name)에 의존하기 때문에 Weak entity이다. 특이한 점은 foreign key로 가져올 때 cur_location과 heading_to라는 두 가지 속성으로 참조해온다는 것이다. Cur_location은 현재 위치, heading_to는 다음으로 향하는 위치를 저장한다. 결국 이 역시 location의 primary key가 foreign key로 Delivery_Info의 primary key를 구성하고 있으므로 identifying으로 타입이 설정되어 있다. Cardinality를 살펴보면, Location 1개가 Delivery_Info의 튜플에 하나도 적용되지 않거나 하나 혹은 여러 개에 적용될 수 있기 때문에 one-to-zero, one or more로 설정되었다.

3.1 Type I

Type1에서 사용되는 Query는 다음의 요구 사항을 충족해야 한다.

Assume truck 1721 is destroyed in a crash. Find all customers who had a package on the truck at the time of the crash. Find all recipients who had a package on that truck at the time of the crash. Find the last successful delivery by that truck prior to the crash.

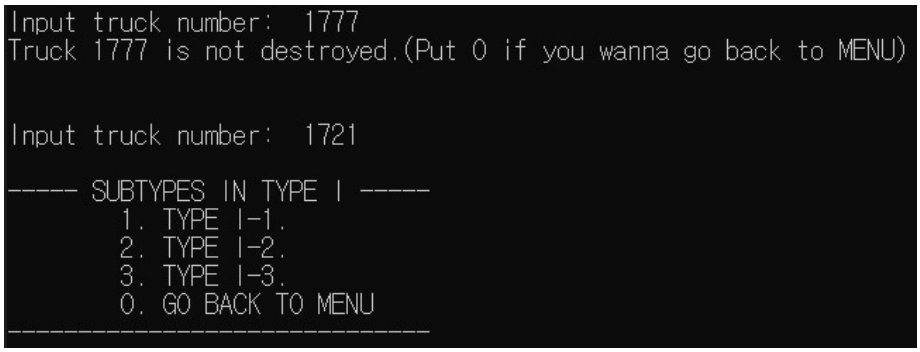
이러한 요구 사항을 충족하기 위해서 3가지의 세부 문제(Sub-type)으로 분리할 수 있다. 첫 번째는 truck 1721이 사고가 난 상황에서 해당 사고에 포함된 패키지의 고객을 출력하라는 것, 두 번째는 수령인을 출력하라는 것, 그리고 세 번째는 같은 트럭에서 마지막으로 성공적으로 배송된 것을 찾으라는 것이다.

1) Type Ⅰ-1

우선 첫 번째, 고객을 찾는 것은 다음과 같은 query를 통해서 가능하다

```
With A(package_ID) as
(select package_ID
From delivery_info
Where vehicle_ID = '1721' and delivery_status = 'crash'),
B(customer_ID) as
(Select distinct customer_ID
From package, A
Where package.package_ID = A.package_ID)
Select customer.customer_ID, customer.name, customer.birth, customer.phone_num
From customer, B
Where customer.customer_ID = B.customer_ID;
```

With를 사용하여 temporary relation을 만드는 방식으로 Query의 구조를 비교적 명료하게 하기 위해 노력하였다. 사용된 임시 릴레이션은 A와 B로 2개가 있으며 A에서는 delivery_info 엔티티에서 사고가 난 트럭에 들어 있는 package_ID 만을 추출한다. B에서는 A와 package를 조인하여 해당 package를 주문한 customer의 ID를 추출하고 최종적으로 customer와 B를 조인하여 customer의 정보를 검색하였다.



우선 truck number를 입력 받아서 만약 해당 트럭이 crash되지 않았다면 에러 메시지를 출력하고 메뉴로 돌아가기 위해서는 0을 입력해야 한다는 정보를 알려주며 다시 truck number를 입력 받는 상황으로 넘어간다. 만약 crash된 truck number를 제대로 입력하게 되면 Subtype들의 메뉴를 출력한다. 여기서 0을 입력하면 다시 기존의 type 1~5까지 포함하는 초기 메뉴로 돌아갈 수 있도록 하였다.

```
---------- TYPE I-1 ----------
** Find all customers who had a package on the truck at the time of the crash **
Customer_ID     Name       Birth           Phone Number
c-0001          Nam        1998-12-29      01011112222
c-0002          Park       1976-01-05      01011112223
c-0003          Kim        1956-09-07      01011112224
------------------------------
```

해당 요청을 처리한 결과는 위와 같이 출력된다. Customer를 찾는 것이 목표인데 ID와 이름 그리고 생년월일과 전화번호를 기본적인 정보라 생각하여 이를 customer 엔티티에서 출력하도록 하였다. 출력된 튜플은 3개가 있는데 해당 튜플이 출력된 이유는 DB에 입력된 정보를 살펴보아야 한다.

```
insert into Delivery_Info values ('p-0001', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0002', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0003', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0004', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
```

우선 Delivery_Info에서 truck number인 1721이며 delivery_status가 crash인 데이터는 위의 총 4개이다. 4개의 데이터는 package 1, 2, 3, 4 네 개가 해당 truck에 있었음을 알려준다.

```
insert into Package values ('p-0001', 'envelope', '1.5', '2023-06-15 12:00:00', 'NO', 'NO', 'Nam', 'a-street', '01011112222', '2023-06-01 12:00:00', 'c-0001');
insert into Package values ('p-0002', 'envelope', '2.5', '2023-06-17 12:00:00', 'NO', 'NO', 'Park', 'b-street', '01011112223', '2023-06-02 12:00:00', 'c-0002');
insert into Package values ('p-0003', 'small box', '3.8', '2023-06-20 12:00:00', 'NO', 'NO', 'Nam', 'a-street', '01011112222', '2023-06-01 12:00:00', 'c-0001');
insert into Package values ('p-0004', 'small box', '4.6', '2023-06-23 12:00:00', 'NO', 'NO', 'Kim', 'c-street', '01011112224', '2023-06-02 12:00:00', 'c-0003');
```

이를 Package 엔티티에 가서 확인해보면 package 1, 2, 3, 4는 customer_ID가 1, 2, 3인 사람이 주문했다는 것을 알 수 있다.

```
insert into Customer values ('c-0001', 'Nam', 'male', '1998-12-29 12:00:00', '01011112222', 'a-street', 'a@gmail.com', 's-0001');
insert into Customer values ('c-0002', 'Park', 'female', '1976-01-05 12:00:00', '01011112223', 'b-street', 'b@gmail.com', 's-0002');
insert into Customer values ('c-0003', 'Kim', 'male', '1956-09-07 12:00:00', '01011112224', 'c-street', 'c@naver.com', 's-0001');
```

이를 다시 customer 엔티티에 가서 확인해보면 Nam, Park, Kim인 것을 알 수 있고 따라서 Query의 결과로 3명의 기본 정보를 customer 엔티티에서 가져와서 출력한 것이다.

2) Type Ⅰ-2

두 번째 수령인을 찾는 것도 유사한 원리로 수행되지만 customer의 정보를 구하는 것은 customer와의 join도 필요하여 보다 복잡한 query를 가졌지만 수령인에 대한 정보는 package 엔티티에 이미 포함되어 있기 때문에 비교적 간소한 구조의 query를 통해서 가능하다

```
select distinct recipient_name, recipient_address, recipient_phone_num
From package, (select package_ID
               From delivery_info
               Where vehicle_ID = '1721' and delivery_status = 'crash') as A
Where package.package_ID = A.package_ID;
```

Package와 delivery_info에서 crash된 truck number에 해당하는 튜플의 package_ID를 A 릴레이션으로 가져오고 이를 package와 조인하는데 ID가 동일한 튜플만을 제한하여 해당 튜플들에 저장된 recipient name과 address, phone number를 출력하는 Query이다.

```
insert into Delivery_Info values ('p-0001', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0002', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0003', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0004', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
```

Delivery Info 엔티티에서 고장난 트럭에 실린 package의 ID는 1,2,3,4로 총 4개 있음을 확인할 수 있었고 이와 동일한 ID의 package에 해당하는 튜플을 package 엔티티에서 아래와 같이 가져올 수 있다.

```
insert into Package values ('p-0001', 'envelope', '1.5', '2023-06-15 12:00:00', 'NO', 'NO', 'Nam', 'a-street', '01011112222', '2023-06-01 12:00:00', 'c-0001');
insert into Package values ('p-0002', 'envelope', '2.5', '2023-06-17 12:00:00', 'NO', 'NO', 'Park', 'b-street', '01011112223', '2023-06-02 12:00:00', 'c-0002');
insert into Package values ('p-0003', 'small box', '3.8', '2023-06-20 12:00:00', 'NO', 'NO', 'Nam', 'a-street', '01011112222', '2023-06-01 12:00:00', 'c-0001');
insert into Package values ('p-0004', 'small box', '4.6', '2023-06-23 12:00:00', 'NO', 'NO', 'Kim', 'c-street', '01011112224', '2023-06-02 12:00:00', 'c-0003');
```

이 때 튜플은 총 4개이지만 package 1과 package3은 동일한 수령인(Nam)을 가지고 있기 때문에 출력 결과에는 Nam, Park, Kim만 출력된다. 출력 결과는 아래와 같다.

```
---------- TYPE I-2 ----------
** Find all recipients who had a package on that truck at the time of the crash **
Name        Address         Phone Number
Nam         a-street        01011112222
Park        b-street        01011112223
Kim         c-street        01011112224
------------------------------
```

3) Type Ⅰ-3

세 번째로 마지막으로 성공된 배달을 찾으라는 query는 다음을 통해서 가능하다.

```
select A.package_ID
from delivery_info as A, (select max(cur_time) as cur_time
                          from Delivery_Info
                          where vehicle_ID = '1721' and delivery_status = 'done') as B
where B.cur_time = A.cur_time and vehicle_ID = '1721' and delivery_status = 'done';
```

우선 1-1, 1-2와 유사하게 delivery_info의 엔티티에서 입력 받은 트럭에 해당하는 튜플들 중에서 delivery_status가 done, 즉 완료가 된 것들의 튜플에서 cur_time이 최대인 튜플을 max를 사용하

여 B라는 릴레이션으로 가져온다. 이것을 다시 Delivery_Info 엔티티와 조인하여 동일한 조건의 튜플들 중 cur_time이 최대인 튜플과 동일한 튜플의 package_ID를 추출하여 출력한다.

```
insert into Delivery_Info values ('p-0001', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0002', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0003', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0004', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0005', '1721', 'Seoul', 'Daegu', 'done', '2022-06-11 12:00:00');
insert into Delivery_Info values ('p-0006', '1721', 'Seoul', 'Daegu', 'done', '2022-06-10 12:00:00');
insert into Delivery_Info values ('p-0007', '1755', 'Daegu', 'Seoul', 'delay', '2022-06-25 12:00:00');
insert into Delivery_Info values ('p-0008', '1755', 'Daegu', 'Seoul', 'delay', '2022-06-25 12:00:00');
insert into Delivery_Info values ('p-0009', '1721', 'Daegu', 'Seoul', 'done', '2021-06-22 12:00:00');
```

1721 트럭을 통해 delivery_info에 저장되어 있는 튜플들은 위와 같다. 2021년에 배달되었던 package9은 done으로 완료되었다는 것을 확인할 수 있고 package5와 6은 2022년에 배달 완료된 것을 확인할 수 있다. 앞선 type 1-1, 1-2에서 확인할 수 있었던 crash된 경우에 담겨 있던 package는 1, 2, 3, 4이다. 이렇다면 truck 1721을 통해 완료된 튜플은 package4, 5, 9가 있으며 해당 튜플들의 cur_time을 본다면 9는 2021년이라 가장 최근이 아니고 4와 5는 2022년이라 최근이지만 4가 6월 11일로 5의 6월 10일보다 하루 더 나중에 배송되었기 때문에 crash된 truck 1721로 가장 최근에 배달되었던 package는 p-0005이다. 출력은 아래와 같다.

```
---------- TYPE 1-3 ----------
** Find the last successful delivery by that truck prior to the crash **
p-0005
------------------------------
```

4) Code Implementation

```
switch (command) {
    case 1:
        while (1) {
            printf("₩nInput truck number:  ");
            scanf("%d", &truck);
            if (truck == 1721 || truck == 0) break;
            else printf("Truck %d is not destroyed.(Put 0 if you wanna go back to MENU)₩n₩n", truck);
        }
        if (truck == 1721) {

            while (1) {
                int back = 0;
                printf("₩n----- SUBTYPES IN TYPE 1 -----₩n");
                printf("₩t1. TYPE 1-1.₩n");
                printf("₩t2. TYPE 1-2.₩n");
                printf("₩t3. TYPE 1-3.₩n");
                printf("₩t0. GO BACK TO MENU₩n");
                printf("------------------------------₩n");
                printf("COMMAND: ");
                scanf("%d", &sub_command);

                switch (sub_command) {
                    case 0:
                        back = 1;
                        break;
                    case 1:
                        printf("₩n---------- TYPE 1-1 ----------₩n");
                        printf("** Find all customers who had a package on the truck at the time of th
                        strcpy(query, "With A(package_ID) as (select package_ID From delivery_info Whe
                        state = mysql_query(connection, query);
```

Type1의 코드를 구현하는데 있어서 설명할 정도로 특이한 부분은 switch와 while의 중첩이다. 우선 메인 메뉴를 출력하는 구조가 while(1)을 통해 무한 루프를 돌며 0이 입력되었을 때만 종료되는 구조를 갖는다. 그리고 주어진 번호에 따라 어떤 TYPE의 QUERY를 실행하는지로 이동하는 switch 문을 가지고 있다. 이 때 유일하게 Type1만 sub type이 있기 때문에 내부적으로 while loop과 switch의 구조를 한 번 더 갖는다. 우선 처음에는 crash된 truck의 번호가 입력되거나 0이 입력 되었을 때만 sub type에 대한 메뉴 창을 띄울 수 있도록 무한 루프를 돌고 만약 고장난 트럭의 번호로 가정한 1721이 들어왔을 때 세부 창을 출력하고 입력을 받아 세부 타입의 명령들을 수행하는 while loop – switch 구문의 구조를 갖는다.

```
strcpy(query, "With A(package_ID) as (select package_ID From delivery_info Where vehicle_ID =
state = mysql_query(connection, query);
if (state == 0) {
    sql_result = mysql_store_result(connection);
    printf("Customer_ID     Name       Birth          Phone Number₩n");
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        printf("%-15s %-10s %-15s %-15s₩n", sql_row[0], sql_row[1], sql_row[2], sql_row[3]);
    }
    mysql_free_result(sql_result);
}
else { //QUERY가 제대로 작동하지 않은 상태에 대한 예외 처리
    fprintf(stderr, "QUERY ERROR₩n");
    exit(0);
}
```

위의 코드는 모든 Query에서 동일하게 사용되는 구조이며 이번에만 이야기하고 뒤에서는 동일하기 때문에 생략하도록 한다. Query에 strcpy를 사용하여 query를 복사하고 mysql_query 함수를 통해 해당 query를 connection된 DB에 보낸다. State는 query가 제대로 수행되었는지를 확인하는 변수인데 0이면 query가 제대로 실행된 것이고 아니면 오류가 발생한 것이다. Query가 제대로 작동하지 않은 것에 대해서 예외 처리를 하였고 제대로 query가 작동하여 state가 0인 경우에는 mysql_store_result로 sql_result 변수에 결과를 저장하고 mysql_fetch_row로 NULL이 나올 때까지 입력 받은 결과에서 튜플을 분리하여 출력을 한다. 모두 출력한 이후에는 mysql_free_result 함수를 사용하여 결과를 저장했던 변수를 초기화한다.

3.2 TypeⅡ

Type2에서는 다음과 같은 요구 사항을 만족해야 한다.

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

```
with Bill_year(c_ID, p_ID) as
(select customer_ID, package_ID
from Bill
where due_date like '2022%'),
A(c_ID, cnt) as
(select c_ID, count(p_ID)
from Bill_year
group by c_ID)
select distinct customer.customer_ID, customer.name, customer.birth, customer.phone_num, B.cnt
from customer, (select c_ID, cnt
                from A
                where cnt = (select max(cnt) from A)) as B
where B.c_ID = customer_ID;
```

이를 구현하는데도 with를 이용하여 2개의 temporary relation을 설정한다. 우선 Bill_year는 특정 년도에 주문된 package와 이를 주문한 customer의 ID를 추출하고 이를 바탕으로 A에서 customer를 기준으로 그룹을 만들었을 때 주문한 package의 개수를 count를 사용하여 추출한다. 이를 바탕으로 추출한 것들 중 max count를 갖는 튜플에 저장된 customer_ID를 A에서 가져와서 customer 엔티티와 조인하여 해당 customer의 정보를 추출하여 출력한다. 이 때 해당 customer가 주문한 package의 개수도 함께 출력한다.

```
insert into Package values ('p-0001', 'envelope', '1.5', '2023-06-15 12:00:00', 'NO', 'NO', 'Nam', 'a-street', '01011112222', '2023-06-01 12:00:00', 'c-0001');
insert into Package values ('p-0002', 'envelope', '2.5', '2023-06-17 12:00:00', 'NO', 'NO', 'Park', 'b-street', '01011112223', '2023-06-02 12:00:00', 'c-0002');
insert into Package values ('p-0003', 'small box', '3.8', '2023-06-20 12:00:00', 'NO', 'NO', 'Nam', 'a-street', '01011112222', '2023-06-01 12:00:00', 'c-0001');
insert into Package values ('p-0004', 'small box', '4.6', '2023-06-23 12:00:00', 'NO', 'NO', 'Kim', 'c-street', '01011112224', '2023-06-02 12:00:00', 'c-0003');
insert into Package values ('p-0005', 'large box', '12.3', '2022-06-16 12:00:00', 'NO', 'NO', 'Kim', 'c-street', '01011112224', '2022-06-01 12:00:00', 'c-0003');
insert into Package values ('p-0006', 'envelope', '3.5', '2022-06-12 12:00:00', 'NO', 'NO', 'Na', 'd-street', '01011112225', '2022-06-03 12:00:00', 'c-0004');
insert into Package values ('p-0007', 'envelope', '1.78', '2022-06-14 12:00:00', 'NO', 'NO', 'Byung', 'e-street', '01011112226', '2022-06-05 12:00:00', 'c-0005');
insert into Package values ('p-0008', 'small box', '5.8', '2022-06-21 12:00:00', 'YES', 'NO', 'Byung', 'e-street', '01011112226', '2022-06-04 12:00:00', 'c-0005');
insert into Package values ('p-0009', 'large box', '13.7', '2021-06-24 12:00:00', 'YES', 'NO', 'Song', 'g-street', '01011112228', '2021-06-03 12:00:00', 'c-0007');
insert into Package values ('p-0010', 'small box', '4.84', '2021-06-30 12:00:00', 'YES', 'YES', 'Ryu', 'h-street', '01011112229', '2021-06-02 12:00:00', 'c-0008');
insert into Package values ('p-0011', 'envelope', '2.65', '2021-06-12 12:00:00', 'NO', 'NO', 'Lee', 'i-street', '01011112230', '2021-06-01 12:00:00', 'c-0009');
insert into Package values ('p-0012', 'large box', '17.5', '2021-06-10 12:00:00', 'NO', 'NO', 'Lee', 'i-street', '01011112230', '2021-06-04 12:00:00', 'c-0009');
```

위는 package 엔티티에 저장된 모든 정보를 가져온 것이다. 위에서부터 package_ID가 1씩 증가하면 총 12개의 튜플로 구성되어 있는데 4번째 속성을 보면 위에서부터 4개씩 2023년, 2022년, 2021년에 주문하고 배송된 상품이라는 것을 알 수 있다. 2023년에는 c-0001에 해당하는 Nam이 2개를 주문하여 가장 많이 주문한 고객이 되며 2022년에는 c-0005에 해당하는 Byung이 가장 많이 주문한 고객이 되고 2021년에는 c-0009에 해당하는 Lee가 될 것이다.

```
---------- TYPE II ----------
** Find the customer who has shipped the most packages in certain year **
(If you wanna go back to MENU, Put 0 as input)

Which Year? : 2020
No data for 2020 year

Which Year? : 2023

Customer_ID     Name       Birth          Phone Number    Num of Packages
c-0001          Nam        1998-12-29     01011112222     2

Which Year? : 2022

Customer_ID     Name       Birth          Phone Number    Num of Packages
c-0005          Byung      2003-12-05     01011112226     2

Which Year? : 2021

Customer_ID     Name       Birth          Phone Number    Num of Packages
c-0009          Lee        1966-02-02     01011112230     2

Which Year? : 0
-----------------------------
```

일단 초기 메뉴에서 2를 눌러 Type2를 들어오면 어떤 내용인지 알려주는 내용을 출력하고 년도를 입력 받는다. 이 때 입력된 연도에 대해 해당하는 고객이 없다면 데이터가 없다는 에러 메시지를 출력하고 다시 년도를 입력 받는다. 2023을 입력했을 때 위에서 예상했듯 Nam이 결과로 출력되고 Num of Packages에 몇 개의 package로 가장 많이 주문한 고객이 되었는지에 대한 정보도 출력된다. 2022와 2021도 예상과 동일한 결과가 출력되었으며 년도를 입력하라는 것에 대해 0을 입력하면 다시 초기 메뉴창으로 이동하도록 설계되었다.

```
printf("₩nWhich Year? : ");
scanf("%d", &year);
if (year == 0) break; //0이 입력일 때만 while loop을 빠져나가도록
sprintf(query, "with Bill_year(c_ID, p_ID) as (select customer_ID, package_ID from Bill where due_date like '%%%d%%')
```

처음에 Query에서는 "due_date like '2022%'"라고 적었지만 그것은 past year이라는 특정 년도에 대해서만 결과를 추출할 수 있는 query이다. 하지만 이번 프로젝트에서는 past year이 아니라 certain year에 대해서 결과를 출력할 수 있어야 하기 때문에 while loop을 돌면서 year을 입력받는다. 따라서 코드를 구현할 때 year == 0일 때만 break를 통해 while loop을 나가 초기 메뉴로 돌아갈 수 있도록 하고 아닌 경우에는 입력 받은 년도를 넣어서 query를 DB에 보내야 한다. 이 때 year이 변수이므로 query를 미리 정해놓을 수 없고 그렇기 때문에 type 1에서 사용했던 strcpy 하여 보내는 구조가 아닌 sprintf를 활용하여 query 변수에 적절한 질의를 넣어주어야 한다. 여기서는 past year이라고 하였지만 실질적으로 code를 구현한 프로그램에서는 certain year를 입력 받아서 해당 년도에 가장 많은 pacakge를 주문한 customer를 찾는다.

```
sql_result = mysql_store_result(connection);
if ((sql_row = mysql_fetch_row(sql_result)) == NULL) { //Query에 해당하는 결과가 없을 때
    printf("No data for %d year₩n", year);
    mysql_free_result(sql_result);
    continue; //다른 year를 받기 위해 while loop을 다시 돈다
}

else { //Query에 해당하는 결과가 있을 때
    mysql_query(connection, query);
    sql_result = mysql_store_result(connection);
    printf("₩nCustomer_ID     Name       Birth          Phone Number    Num of Packages₩n");
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        int count = atoi(sql_row[4]);
        printf("%-15s %-10s %-15s %-15s %-10d₩n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], count);
    }
    mysql_free_result(sql_result);
    continue; //출력을 완료 했으면 다시 year을 받기 위해 while loop을 돈다
}
```

또한 해당 년도에 대한 결과가 있으면 type1에서 설명한 것과 동일한 원리로 출력하면 되지만 그렇지 않은 경우 데이터가 없고 다시 년도를 입력 받을 수 있는 것을 구현해야 한다. 따라서 while 문을 사용하지 않으면서 mysql_fetch_row를 한 번만 수행하고 해당 값이 NULL인지 확인해본다. NULL이라면 query에서 해당하는 결과에 대한 튜플이 하나도 없다는 것이고 그런 경우 다른 year를 받기 위해 while loop을 돌 수 있도록 continue를 실행하도록 코드를 작성했다.

3.3 TypeⅢ

Type2에서는 다음과 같은 요구 사항을 만족해야 한다. 3번 query는 2번 query와 유사하지만 money를 계산하는 것이 달라졌다.

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

```
with Bill_year(c_ID, charge) as
(select customer_ID, charge
from Bill
where due_date like '2022%'),
A(c_ID, sum) as
(select c_ID, sum(charge)
from Bill_year
group by c_ID)
select distinct customer.customer_ID, customer.name, customer.birth, customer.phone_num, B.sum
from customer, (select c_ID, sum
                from A
                where sum = (select max(sum) from A)) as B
where B.c_ID = customer_ID;
```

Type2와 동일하게 Bill_year, A, B라는 릴레이션을 임시적으로 만드는 방식으로 query가 구성되어 있다. 유일한 차이점은 Bill로부터 charge를 가져오고 A에서는 charge의 sum을 사용하는 것이다. Type2에서는 count를 사용했다면 이번 Type3에서는 money를 구해야 하기 때문에 sum이 사용되었다.

```
insert into Bill values ('3000', '2023-06-03 12:00:00', 'cash', '10001000', 'prepaid', '2023-06-02 12:00:00', 'p-0001', 'c-0001');
insert into Bill values ('4000', '2023-06-04 12:00:00', 'cash', '10001001', 'prepaid', '2023-06-03 12:00:00', 'p-0002', 'c-0002');
insert into Bill values ('7000', '2023-06-05 12:00:00', 'transfer', '10001000', 'prepaid', '2023-06-03 12:00:00', 'p-0003', 'c-0001');
insert into Bill values ('6000', '2023-06-05 12:00:00', 'credit card', '10001002', 'monthly', '2023-06-03 12:00:00', 'p-0004', 'c-0003');
insert into Bill values ('20000', '2022-06-05 12:00:00', 'credit card', '10001002', 'monthly', '2022-06-03 12:00:00', 'p-0005', 'c-0003');
insert into Bill values ('3000', '2022-06-05 12:00:00', 'transfer', '10001005', 'prepaid', '2022-06-04 12:00:00', 'p-0006', 'c-0004');
insert into Bill values ('2000', '2022-06-07 12:00:00', 'credit card', '10002000', 'monthly', '2022-06-06 12:00:00', 'p-0007', 'c-0005');
insert into Bill values ('6000', '2022-06-07 12:00:00', 'credit card', '10002000', 'monthly', '2022-06-06 12:00:00', 'p-0008', 'c-0005');
insert into Bill values ('3000', '2021-06-07 12:00:00', 'cash', '10001030', 'prepaid', '2021-06-05 12:00:00', 'p-0009', 'c-0007');
insert into Bill values ('3000', '2021-06-07 12:00:00', 'cash', '10001020', 'prepaid', '2021-06-05 12:00:00', 'p-0010', 'c-0008');
insert into Bill values ('3000', '2021-06-07 12:00:00', 'credit card', '10003000', 'monthly', '2021-06-05 12:00:00', 'p-0011', 'c-0009');
insert into Bill values ('3000', '2021-06-07 12:00:00', 'credit card', '10003000', 'monthly', '2021-06-05 12:00:00', 'p-0012', 'c-0009');
```

Bill 엔티티에 저장된 정보를 본다면 위에서부터 4개씩 2023년, 2022년, 2021년에 해당하는 정보라는 것을 확인할 수 있다. Type3에서는 개수가 중요했지만 이번 Type4에서는 첫 번째 속성에 해당하는 charge가 가장 중요하다. 물론 여러 개의 package를 주문했다면 charge가 누적되어 가장 많은 돈을 쓴 고객으로 나올 수도 잇겠지만 주문한 package에 부과된 요금이 크다면 주문한 package가 많지 않더라도 이번 type의 출력 결과가 될 수 있다. 2023년에는 c-0001에 해당하는 Nam이 가장 많은 주문을 하였고 charge의 합계도 10000으로 가장 높다. 하지만 2022년에는 c-0005에 해당하는 Byung이 2개로 가장 많이 주문했지만 1개를 주문한 c-0003의 Kim이 20000으로 더 높은 charge를 낸 것을 확인할 수 있다. Byung의 두 개의 package에 대한 charge의 합계

가 8000이기 때문에 오히려 1개를 주문한 Kim이 더 높은 돈을 낸 고객으로 출력될 것이다.

```
---------- TYPE III ----------
** Find the customer who has shipped the most money in certain year **
(If you wanna go back to MENU, Put 0 as input)

Which Year? : 2000
No data for 2000 year

Which Year? : 2023

Customer_ID     Name       Birth           Phone Number    Money Spent
c-0001          Nam        1998-12-29      01011112222     10000.000

Which Year? : 2022

Customer_ID     Name       Birth           Phone Number    Money Spent
c-0003          Kim        1956-09-07      01011112224     20000.000

Which Year? : 2021

Customer_ID     Name       Birth           Phone Number    Money Spent
c-0009          Lee        1966-02-02      01011112230     6000.000

Which Year? : 0
------------------------------
```

위의 사진이 프로그램에서 결과를 실행한 것이며 이 역시 정보가 없는 년도에 대해서는 데이터가 없다는 오류 메시지를 출력하고 다시 년도를 입력받는다. 각 년도에 대해 가장 많은 돈을 지불한 고객의 정보를 customer 엔티티와 조인하여 기본 정보를 출력하였고 마지막 속성으로 해당 년도에 사용한 금액의 합계를 출력하였다.

Code implementation에 대해서는 Type3가 Type2와 완전히 동일한 구조를 갖고 있기 때문에 이에 대한 부분은 생략하도록 하겠다.

3.4 TypeIV

Type4에서는 다음과 같은 요구 사항을 만족해야 한다.

"Find those packages that were not delivered within the promised time."

```
with A(package_ID) as
(select distinct package_ID
from Delivery_Info
where delivery_status = 'delay' or delivery_status = 'crash')
select B.package_ID, B.package_type, B.weight, B.estimated_date
from package as B, A
where A.package_ID = B.package_ID
```

요구 사항을 만족시키기 위해 설계한 Query는 위와 같다. With를 사용하여 delivery_info에서 delay되거나 crash된 튜플의 package_ID를 가져온다. 이 때 distinct를 사용하여야만 한다. 왜냐하면 package_ID는 package 엔티티에서는 primary key로 중복되는 정보가 없는 것이 보장되지만 delivery_info에서는 primary key를 구성하는 속성 중 하나이기 때문에 중복이 없다는 보장이 되지 않기 때문이다. 실제로도 Delivery Info에는 동일한 패키지가 이동하는 것에 따라 다른 시간에 다른 장소에서 로그가 찍히기 때문에 distinct를 통해 중복되는 package_ID는 배제하여야 한다. 이렇게 가져온 package_ID를 package 엔티티와 조인하여 해당 package의 기본 정보들을 추출받아서 출력할 수 있도록 query가 설계되었다.

```
insert into Delivery_Info values ('p-0001', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0002', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0003', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0004', '1721', 'Seoul', 'Daegu', 'crash', '2023-06-13 12:00:00');
insert into Delivery_Info values ('p-0005', '1721', 'Seoul', 'Daegu', 'done', '2022-06-11 12:00:00');
insert into Delivery_Info values ('p-0006', '1721', 'Seoul', 'Daegu', 'done', '2022-06-10 12:00:00');
insert into Delivery_Info values ('p-0007', '1755', 'Daegu', 'Seoul', 'delay', '2022-06-25 12:00:00');
insert into Delivery_Info values ('p-0008', '1755', 'Daegu', 'Seoul', 'delay', '2022-06-25 12:00:00');
insert into Delivery_Info values ('p-0009', '1721', 'Daegu', 'Seoul', 'done', '2021-06-22 12:00:00');
insert into Delivery_Info values ('p-0010', '1600', 'Seoul', 'Texas', 'delay', '2021-07-10 12:00:00');
```

Delivery_Info에 저장된 튜플들을 살펴보면 p-0001, 2, 3, 4는 crash로 promised time보다 늦어졌으며 p-0007, 8, 10은 delay되었다는 것을 확인할 수 있다. 따라서 출력 결과로 p-0001, 2, 3, 4, 7, 8, 10에 해당하는 7개의 패키지에 대한 정보가 출력되어야 할 것이다.

```
---------- TYPE IV ----------
** Find the packages that were not delivered within the promised time **
Package_ID      Package_Type    Weight          Estimated Date
p-0001          envelope        1.500           2023-06-15
p-0002          envelope        2.500           2023-06-17
p-0003          small box       3.800           2023-06-20
p-0004          small box       4.600           2023-06-23
p-0007          envelope        1.780           2022-06-14
p-0008          small box       5.800           2022-06-21
p-0010          small box       4.840           2021-06-30
-----------------------------
```

Type4의 경우 따로 input으로 입력 받는 것이 없기 때문에 type1에서 사용했던 것처럼 strcpy를 통해 기존에 설정되어 있는 query를 DB에 보내고 결과물을 출력하는 방식으로 코드가 설계되었다. 또한 query를 수행하면 결과를 출력하고 바로 초기 메뉴로 이동할 수 있도록 설계했다.

3.5 Type V

Type5에서는 다음과 같은 요구 사항을 만족해야 한다.

Generate the bill for each customer for the past month. Consider creating several types of bills.

Type5에서는 customer에 대한 Bill을 만드는 것이 핵심이고 세부적인 내용은 자세히 나오지 않았기 때문에 어떤 정보가 Bill에 포함될지는 임의적으로 선택하였다.

```
case 5:
    printf("₩n---------- TYPE V ----------₩n");
    printf("** Generate the Bill for each customer for the certain month **₩n");
    printf("(If you wanna go back to MENU, Put 0 as input)₩n");
    while (1) {
        printf("Which year: ");
        scanf("%d", &year);
        if (year == 0) break; //0을 입력 받으면 초기 메뉴로 돌아가도록
        printf("Which month: ");
        scanf("%d", &month);
        if (month == 0) break; //0을 입력 받으면 초기 메뉴로 돌아가도록
```

우선 Bill을 설계하기 위해서 주어지는 input으로는 year과 month이며 해당 year과 month에 관련된 튜플이 없다면 오류 메시지를 출력하고 다시 year와 month를 입력하도록 설계하였다. 그러다가 year과 month 중 아무거나 0을 입력하면 초기 메뉴 화면으로 돌아갈 수 있도록 코드를 구현했다.

```
select customer.name, customer.customer_ID
from customer, (select distinct customer_ID
                from Bill
                where due_date like '%%%d-0%d%') as A
where A.customer_ID = customer.customer_ID;
```

다음으로는 주어진 year와 month를 활용하여 query를 작성하여 DB에 query를 보내어 3가지 정보를 저장한다. 첫 번째는 고객의 숫자이고 두 번째는 customer의 이름이며 세 번째는 customer의 ID이다. 고객의 숫자는 이후 Bill을 몇 개 만들어야 하는지 정해야 하기 때문에 따로 C 코드를 작동하는 과정에서 업데이트 하는 것이고 customer의 이름은 Bill에 들어갈 이름을 위해 저장하며 ID는 이후 query에서 사용하기 위해 저장하는 것이다.

다음 단계는 구한 고객의 수만큼 for loop을 돌면서 Bill의 이름을 설정하여 해당 이름으로 txt 파일을 만들고 2가지 query를 DB에 전달하여 2 종류의 정보를 차례대로 txt 파일에 저장하도록 한다. 첫 번째는 Bill의 상단에 customer의 개인 정보를 나타내는 것이고 두 번째는 그 아래에 해당 customer가 주어진 Year-Month에 주문한 package의 정보를 나타내는 것이다. 이는 아래의 Query들을 통해 구현되었다.

```
select *
from customer
where customer_ID = '%s';
```

우선 첫 번째로 Bill의 상단에 저장할 customer의 개인 정보는 customer의 모든 속성을 가지고 오는데 이때 where문에서 사용하는 조건은 customer_ID가 아까 저장했던 customer의 ID와 같은 것을 활용하는 것이다. 그리고 받아온 모든 정보를 활용하지는 않고 필요한 정보들에 해당하는 row를 파일에 fprintf하는 방법을 사용한다.

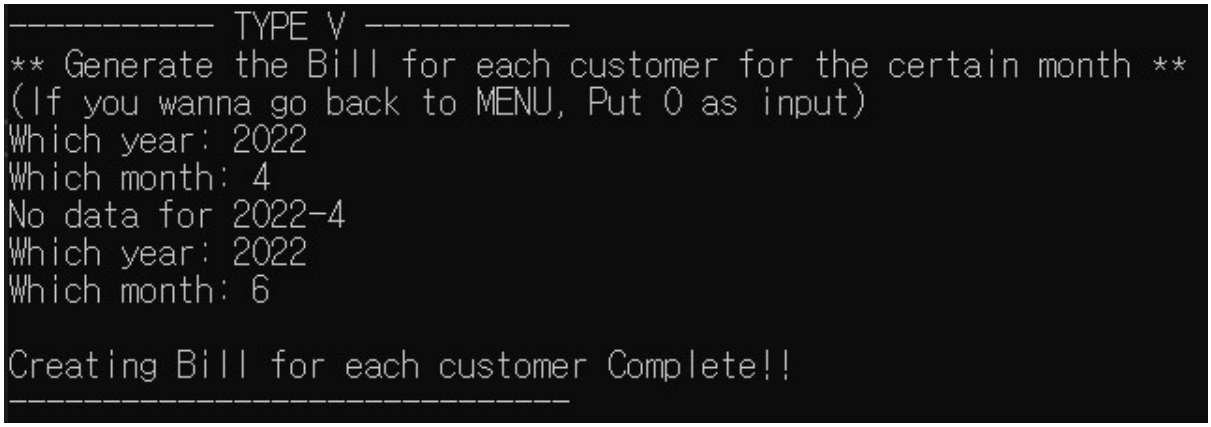
```
with A(p_ID, p_type) as
(select package.package_ID, package_type
from package
where package.customer_ID = '%s')
select p_ID, p_type, Bill.charge, Bill.payment_method, Bill.payment_status, Bill.payment_date
from Bill, A
where Bill.package_ID = p_ID and Bill.due_date like '%%%d-0%d%%';
```

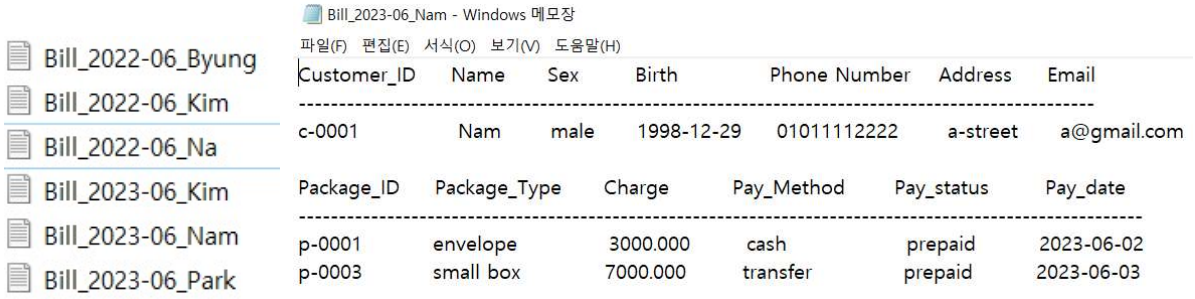
두 번째로 해당 customer가 해당 연월에 주문한 package를 찾는 Query이다. With를 사용하여 A relation을 임시로 만드는데 A relation에는 이미 저장했던 해당 customer의 ID와 매칭되는 package의 ID와 type을 추출한다. 이를 Bill과 조인하여 Bill에서 추가적으로 해당 package에 대해 나타낼 수 있는 정보들을 추가적으로 추출하여 이를 txt 파일에 저장하는 방식으로 구현하였다.

```
for (int i = 0; i < num_of_customer; i++) {
    if(month < 10)
        sprintf(filename, "Bill_%d-0%d_%s.txt", year, month, customer_name[i]);
    else
        sprintf(filename, "Bill_%d-%d_%s.txt", year, month, customer_name[i]); //customer의 이름과 연도, 월을 포함한 Bill의 이름을 설정
    fp = fopen(filename, "w"); //해당 Bill 만들기

    //Bill의 상단에 customer의 개인 정보를 나타냄
    sprintf(query, "select * from customer where customer_ID = '%s'", customer_ID[i]);
    mysql_query(connection, query);
    sql_result = mysql_store_result(connection);

    fprintf(fp, "Customer_ID     Name      Sex       Birth            Phone Number    Address     Email₩n");
    fprintf(fp, "-------------------------------------------------------------------------------------------------------₩n");
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        fprintf(fp, "%-15s     %-8s  %-7s   %-14s %-15s   %-11s   %-15s₩n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4], s
    }
    mysql_free_result(sql_result);

    //해당 customer가 주문한 Package의 정보에 대해 나타냄
    if(month < 10)
        sprintf(query, "with A(p_ID, p_type) as (select package.package_ID, package_type from package where package.customer_ID = '%s') s
    else
        sprintf(query, "with A(p_ID, p_type) as (select package.package_ID, package_type from package where package.customer_ID = '%s') s
    mysql_query(connection, query);
    sql_result = mysql_store_result(connection);

    fprintf(fp, "₩nPackage_ID     Package_Type      Charge        Pay_Method      Pay_status       Pay_date₩n");
    fprintf(fp, "-------------------------------------------------------------------------------------------------------₩n");
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        float charge = atof(sql_row[2]);
        fprintf(fp, "%-15s %-20s %-15.3f %-20s %-15s %-15s₩n", sql_row[0], sql_row[1], charge, sql_row[3], sql_row[4], sql_row[5]);
```

위의 사진은 직접적으로 for loop을 돌면서 Bill을 만들고 query를 2번 실행하여 필요한 정보들을 Bill에 저장하는 과정을 담았다. 한 가지 신경 써야 할 부분은 year과 month를 따로 입력 받기 때

문에 year를 사용하는 것에는 4칸이 고정이라 문제 없지만 month의 경우 'date' 데이터 타입을 가진 경우 06, 07 이런 식으로 저장되기 때문에 이에 맞도록 month가 10보다 작은 경우에는 query에 0을 하나 추가하여 제대로 query가 작동할 수 있도록 하였다.





위의 세 사진은 프로그램을 실행한 결과이며 프로그램에 의해 Bill이 완성되었을 때 해당 폴더에 만들어진 Bill의 리스트, 그리고 실제로 Bill 내부에 적힌 자료들을 보여준다. 우선 프로그램에서 정보가 없는 YEAR-MONTH가 들어오면 데이터가 없다는 오류 메시지를 출력하고 다시 YEAR-MONTH의 조합을 받도록 while loop을 돌게 설계하였다. 그리고 정보가 있는 달의 경우에는 위에서 설명한 Bill을 만드는 과정을 거쳐 문제 없이 완료되었을 때, 이를 알리는 표현을 프로그램 상에서 출력하고 폴더 안에도 txt 파일을 만들 수 있도록 하였다.