

**Database System**

# Package Delivery System

**Project 2. Normalization and Query Processing**

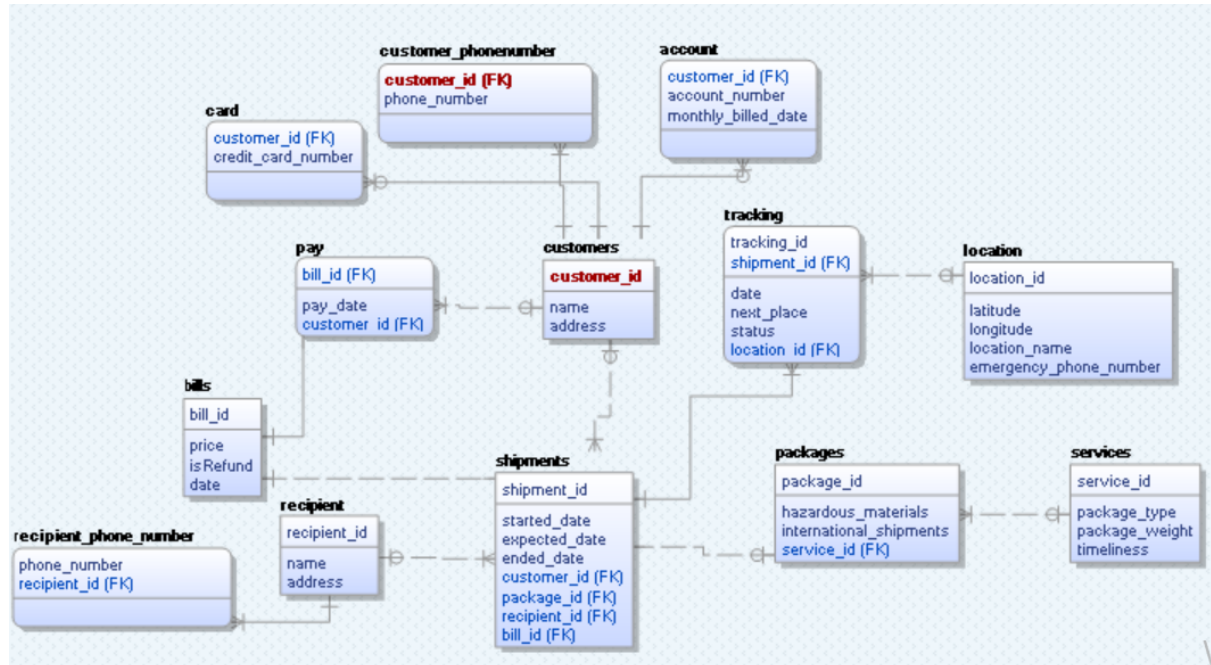
담당 교수 : 정성원

이름 : 김도현

학번 : 20181256

## 1. BCNF Decomposition & detail modified

먼저 필자가 구상한 Proejct1 에서 구상한 Schema Diagram 도표를 사진 첨부하였다. 해당 Schema entity를 하나씩 보며 BCNF가 만족하도록 바꾸도록 하겠다.



<수정 전!!>

- Customers(customer\_id, name, address)
- Shipments(shipment\_id, started\_date, expected\_date, ended\_date, customer\_id(FK), package\_id(FK), recipient\_id(FK), bill\_id(FK))
- Packages(package\_id, hazardous\_materials, international\_shipments, service\_id(FK))
- Services(service\_id, package\_type, package\_weight, timeliness)
- Tracking(tracking\_id, shipment\_id(FK), date, next\_place, status, location\_id(FK))
- Location(location\_id, latitude, longitutde, location\_name, emergency\_phone\_number)
- Recipient(recipient\_id, name, address)
- Recipient\_phone\_number(phone\_number, recipient\_id(FK))
- Customer\_phone\_number(customer\_id(FK), phone\_number)
- Card(customer\_id(FK), credit\_card\_number)
- Account(customer\_id(FK), account\_number, monthly\_billed\_date)
- Pay(bill\_id(FK), pay\_date, customer\_id(FK))
- Bills(bill\_id, price, isRefund, date)

BCNF 를 만족하는 조건은 다음과 같다.

a,b is in R

a->b(Funtional dependency) is trivial

a is a superkey of R

1. Customers(customer\_id, name, address)

Funtional dependency: Customer\_id->name,address

Customer\_id is superkey of Customers

BCNF를 만족한다.

2. Shipments(shipment\_id, started\_date, expected\_date, ended\_date, customer\_id(FK),  
package\_id(FK), recipient\_id(FK), bill\_id(FK))

Funtional dependency: shipment\_id->started\_date, expected\_date, ended\_date, customer\_id,  
package\_id, recipient\_id, bill\_id

Shipment\_id is superkey of Shipments

BCNF를 만족한다.

이때 여기서 bill\_id 가 있다. 하지만 실제로 bill\_id와 shipments는 같은 id로 구성할 수 있다. 왜냐하면 모든 배송에는 하나의 bill이 필요하기 때문이다. 그래서 bill\_id를 해당 속성에서 삭제하고 packages에서 받아온 foreign key 인 package\_id는 추후에 설명하겠지만 price로 package 가격을 지칭하는 것으로 바꿨다.

수정본은 다음과 같다.

Shipments(shipment\_id, started\_date, expected\_date, ended\_date, customer\_id(FK),  
price(FK), recipient\_id(FK))

Funtional dependency: shipment\_id->started\_date, expected\_date, ended\_date, customer\_id, price,  
recipient\_id

Shipment\_id is superkey of Shipments

BCNF를 만족한다.

### 3. Packages(package\_id, hazardous\_materials, international\_shipments, service\_id(FK))

Functional dependency: Package\_id->hazardous\_materials, international\_shipments, service\_id

Package\_id is superkey of Packages

BCNF를 만족한다.

이때 package 에 대해 고려하지 않았던 점이 package 의 가격이 얼마인지 모른다는 것이다. 그래서 package\_id 대신 해당 package의 가격을 입력할 수 있도록 Price 로 이름을 바꿨다.

Functional dependency: price-> hazardous\_materials, international\_shipments, service\_id

Price is superkey of packages.

BCNF를 만족한다.

수정본

Packages(price, hazardous\_materials, international\_shipments, service\_id(FK))

### 4. Tracking(tracking\_id, shipment\_id(FK), date, next\_place, status, location\_id(FK))

여기서 다시 생각해보니 tracking\_id 만으로도 나머지 attribute을 정의할 수 있다고 판단했다. 그래서 tracking\_id 만 primary key로 두고 foreign key 인 shipment\_id는 primary key에서 제외 시켰다.

수정본

Tracking(tracking\_id, shipment\_id(FK), date, next\_place, status, location\_id(FK))

Functional dependency: tracking\_id-> shipment\_id, date, next\_place, status, location\_id

tracking\_id is superkey of Tracking

BCNF를 만족한다.

### 5. Location(location\_id, latitude, longitude, location\_name, emergency\_phone\_number)

Functional dependency: location\_id->latitude, longitude, location\_name, emergency\_phone\_number

location\_id\_id is superkey of Location

BCNF를 만족한다.

6. Recipient(recipient\_id, name, address)

Functional dependency: recipient\_id->name, address

Recipient\_id is superkey of recipient

BCNF를 만족한다.

7. Recipient\_phone\_number(phone\_number, recipient\_id(FK))

Functional dependency: phone\_number, recipient\_id -> phone\_number, recipient\_id

This Fd is trivial.

BCNF를 만족한다.

8. Customer\_phone\_number(customer\_id(FK), phone\_number)

Functional dependency: phone\_number, customer\_id -> phone\_number, customer\_id

This Fd is trivial.

BCNF를 만족한다.

9. Card(customer\_id(FK), credit\_card\_number)

Functional dependency: customer\_id, credit\_card\_number -> customer\_id, credit\_card\_number

Credit\_card\_number->customer\_id

여기서는 customer\_id, credit\_card\_number를 primary key로 두었는데 다시 생각해보니 credit\_card\_number는 모두가 고유하기 때문에 최소성을 갖는 candidate key 속성을 고려하여 credit\_card\_number 만 Primary key로 설정하겠다.

수정 사항: Card(credit\_card\_number, customer\_id(FK))

Functional dependency:

Credit\_card\_number->customer\_id

This Fd is trivial.

BCNF를 만족한다.

10. Account(customer\_id(FK), account\_number, monthly\_billed\_date)

Functional dependency: customer\_id, account\_number, monthly\_billed\_date → customer\_id, account\_number, monthly\_billed\_date

This Fd is trivial.

BCNF를 만족한다.

11. Pay(bill\_id(FK), pay\_date, customer\_id(FK))

Functional dependency: bill\_id → pay\_date, customer\_id

bill\_id is superkey of pay

BCNF를 만족한다.

12. Bills(bill\_id, price, isRefund, date)

Functional dependency: bill\_id → pay\_date, customer\_id

bill\_id is superkey of Bills

BCNF를 만족한다.

이때 다시 고려를 하여 Pay와 Bills를 하나로 합쳐도 문제가 되지 않을 것 같았다. 그리고 shipments 에서 bcnf에 대해 판단할때 알게 된 사실은 bill\_id와 shipment\_id가 동일해야 한다는 것이다. 그래서 shipments로 부터 shipment\_id를 foreign key로 받아 이를 primary key로 설정하기로 했다. 다만 해당 attribute의 이름은 그대로 bill\_id를 사용하였다. 마지막으로 price의 경우 이미 shipments에서 price를 정의하고 있기 때문에 정보의 중복을 제거하기 위해 Price는 생략하였다.

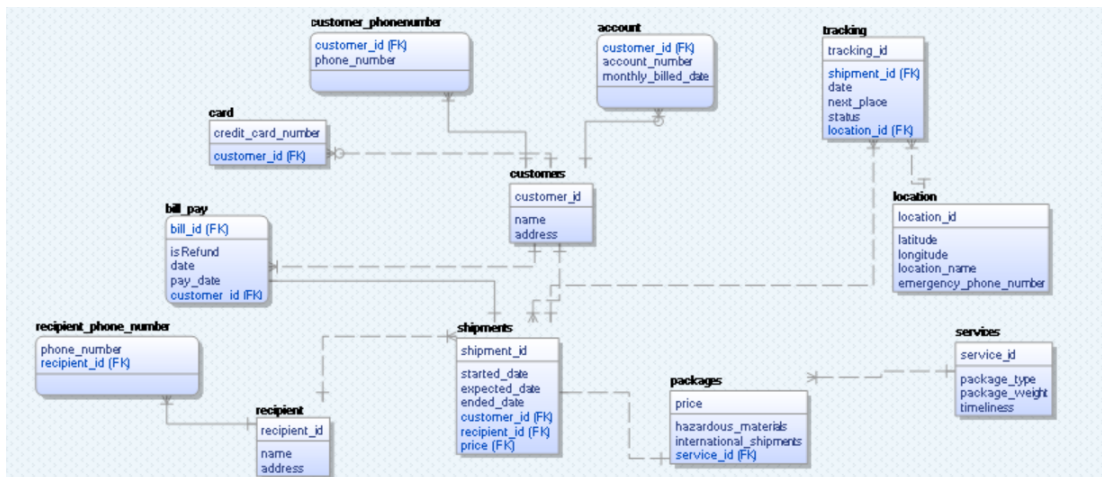
수정 사항: Bill\_Pay(bill\_id, isRefund, date, pay\_date, customer\_id(FK))

Functional dependency: bill\_id → isRefund, date, pay\_date, customer\_id

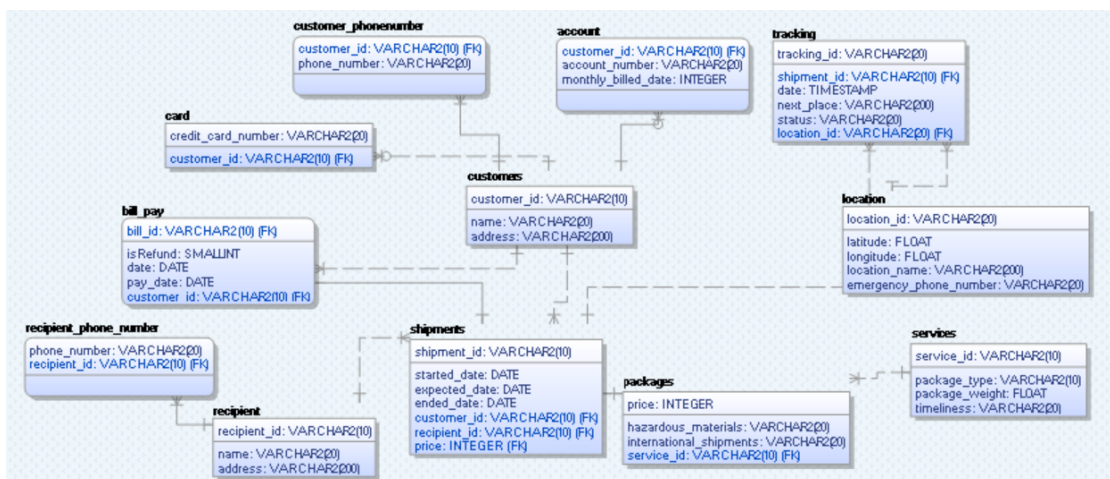
bill\_id is superkey of bill&pay

BCNF를 만족한다.

다음은 수정을 거친 schema diagram 수정본이다.



## 2. Physical Schema diagram



다음은 logical schema 를 Physical schema로 바꾼 것이다. 기본적으로 null을 허용하지 않도록 했지만 일부 attribute의 경우 null을 허용해야만 했다. 세부 table을 하나씩 살펴해보도록 하겠다.

- shipments

The screenshot shows a table named 'shipments' with the following columns and data types: shipment\_id: VARCHAR2(10), started\_date: DATE, expected\_date: DATE, ended\_date: DATE, customer\_id: VARCHAR2(10) (FK), recipient\_id: VARCHAR2(10) (FK), and price: INTEGER (FK).

shipments	
shipment_id:	VARCHAR2(10)
started_date:	DATE
expected_date:	DATE
ended_date:	DATE
customer_id:	VARCHAR2(10) (FK)
recipient_id:	VARCHAR2(10) (FK)
price:	INTEGER (FK)

다음은 Shipments table이다.

예시

('SHIP001', '2022-01-01', '2022-01-03', '2022-01-03', 'CUST001', 'REC001', 7500)

우선 primary key 로 shipment\_id를 가진다. 이는 shipment 를 구분하기 위한 핵심 key이다. Ship001 처럼 varchar을 최대 10개까지 받을 수 있도록 했다. Started\_date 는 배송 출발일을 의미한다. Expected\_Date는 배송 예정 도착일을 의미한다. Ended\_date는 배송 실제 도착일을 의미한다. 여기는 모두 data type을 date를 사용 했다. 이때 started\_Date, expected\_Date는 모두 null을 허용하면 안되지만 ended\_date는 어떤 사고나 아니면 아직 배송이 미완료 된 택배도 있기 때문에 null을 허용했다. Customer\_id는 해당 배송을 보내는 고객을 의미하는 것으로 varchar을 최대 10개까지 허용하고 recipient\_id는 해당 배송을 받는 수령인을 의미 하는 것으로 varchar을 최대 10개까지 허용한다. 그리고 마지막 price는 해당 택배 배송에 대한 가격을 의미한다. Price는 package 별로 고유하기 때문에 packages의 primary key이다. 이들은 모두 Null을 허용하지 않는다.

- recipient

The screenshot shows a table named 'recipient' with the following columns and data types: recipient\_id: VARCHAR2(10), name: VARCHAR2(20), and address: VARCHAR2(200).

recipient	
recipient_id:	VARCHAR2(10)
name:	VARCHAR2(20)
address:	VARCHAR2(200)

예시

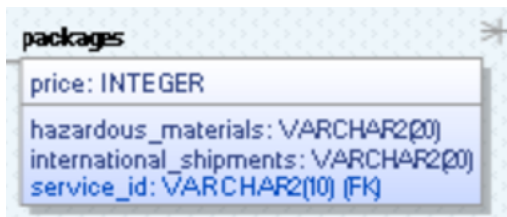
('REC001', 'Jong Sub', '123 Maple Avenue, Gangnam-gu, Seoul, South Korea')

수령인에 대한 정보를 저장하는 테이블이다. Recipient\_id는 수령인 고유 id로



varchar을 최대 10개까지 받을 수 있도록 하였다. 그리고 name은 해당 수령인에 대한 이름이다. 이름은 중복이 될 수 있기 때문에 primary key로 사용하지 않았다. 여기서는 varchar를 사용해 최대 20개까지 받을 수 있도록 하였다. 마지막으로 address는 수령인에 대한 주소를 저장하는 attribute으로 어떤 주소가 올지 모르니 varchar을 최대한 길게 약 200개까지 받을 수 있도록 설정하였다. 이들은 모두 필수 정보이기 때문에 Null을 허용하지 않는다.

#### - Packages

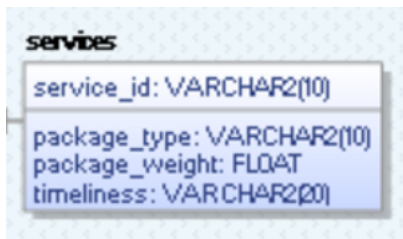


예시

(2500, 'Yes', 'Yes', 'SRV001')

물건의 구성 package를 의미한다. 해당 package는 integer인 price를 기준으로 위험 물질, 해외 배송, 서비스 종류를 알 수 있다. 물건의 id, package\_id를 primary key로 잡았다. 위험물질이 있거나 해외배송이면 “yes” 아니면 “No”가 들어올 수 있도록 default 값인 varchar를 사용했다. Service\_id는 service 종류를 구분할때 확인하는 attribute으로 이 역시 service의 primary key이기 때문에 varchar을 최대 10개까지 허용하도록 했다. 이들 모두 Null을 허용하지 않는다.

#### - Services



예시

('SRV001', 'Standard', 1.5, 'Normal')

배달 서비스에 대한 정보를 나타내는 테이블이다. 우선 `service_id`는 해당 서비스 고유의 primary key이고 `package_type`은 standard, Express, Premium 이렇게 3가지로 나뉘어 timeliness를 정의한다. (Normal, Fast, Very Fast) 그리고 각 type별로 각각 가능한 package 무게가 다르다. 그래서 무게를 소수점까지 받을 수 있도록 float 데이터 타입을 쓰고 나머지는 varchar를 사용해서 해당 정보들을 저장 할 수 있도록 했다. 이들 모두 null을 허용하지 않는다.

#### - Customers



예시

(CUST001', 'DD Kim', '123 Seoul Street, Gangnam-gu, Seoul, South Korea')

고객에 대한 정보로 어떤 고객이 물건을 배송업체에 맡기고 보내는 지에 대한 table이다. `customer_id`는 고객 고유 id로 varchar를 최대 10개까지 받을 수 있도록 하였다. 그리고 `name`은 해당 고객에 대한 이름이다. 이름은 중복이 될 수 있기 때문에 primary key로 사용하지 않았다. 여기서는 varchar를 사용해 최대 20개까지 받을 수 있도록 하였다. 마지막으로 `address`는 고객에 대한 주소를 저장하는 attribute으로 어떤 주소가 올지 모르니 varchar를 최대한 길게 약 200개까지 받을 수 있도록 설정하였다. 이들은 모두 필수 정보이기 때문에 Null을 허용하지 않는다.

#### - Bill\_pay

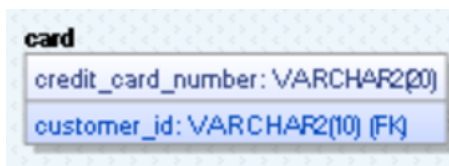


예시

('SHIP001', 0, '2022-01-01', '2022-01-02', 'CUST001')

고객이 택배를 보낸 영수증에 대한 정보를 기록하는 테이블이다. 여기서는 bill\_id 라는 고유 key를 varchar을 사용해서 최대 20개까지 문자열을 받는다. isRefund는 환불 여부로 0이나 1로 구성되어 1이면 환불되었다는 것을 의미한다. Date는 bill 이 생성된 attribute으로 DATE를 data type을 받는다. 이들을 모두 Null을 허용하지 않는 필수 정보들이다. 그리고 pay\_date는 실제 고객이 돈을 지불한 날짜를 의미한다. 돈을 내지 않았을 경우가 있기 때문에 pay\_date는 null을 허용하도록 하였다. 마지막 customer id는 해당 bill이 누구의 고객 것인지 알 수 있는 attribute으로 varchar을 최대 10개까지 문자열을 받을 수 있도록 하였다.

- Card

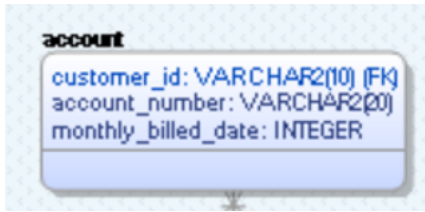


예시

('1234567890123456', 'CUST001')

고객의 카드정보를 저장하는 테이블이다. 이들은 모두 필수적인 요소로 Null을 허용하지 않고 card number같은경우 최대 20자리 까지, customer\_id는 최대 10자리까지 허용하도록 했다.

- Account



여기서는 card entity 와 달리 규칙적으로 거래를 하는 customer 에 대한 계좌 정보를 저장하고 있다. Attribute으로 account\_number, monthly-billed\_Date 를 뒤서 계좌 번호, 그리고 언제 돈이 인출되는지에 대한 날짜 정보를 가진다. 이들을 discriminator 로 사용해서 primary key인 customer\_id와 함께 고유성을 가지도록 한다. 그래서 모두 null이 허용되지 않고 monthly\_billed\_Date 같은 경우 월별 해당하는 달이 정해져 있기 때문에 1~31 사이의 숫자로 들어오면 된다. 그래서 integer 로 설정했다.

#### - Tracking



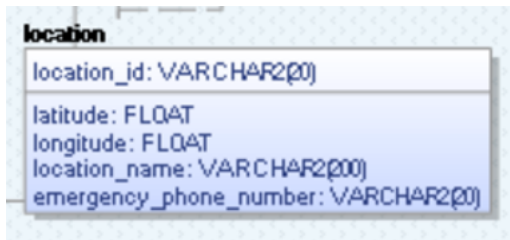
예시

('TRACK001', 'SHIP001', '2022-01-03 10:00:00', 'Main Office', 'stay', 'LOC001')

여기서는 배송정보를 추적하는 테이블이다. 해당 테이블은 tracking\_id를 varchar 문자열 최대 20개까지 받도록 허용한다. 그리고 shipment\_id는 각 택배에 대한 key로 varchar 최대 10개까지 받도록 했다. Date는 검색을 원하는 시간으로 timestamp를 써서 같은 날이더라도 여러개의 시간대가 형성될 수 있도록 하였다. 그리고 next\_place는 이 시간 직후 택배가 있을 장소를 의미한다. 장소 이름이 길 수도 있기 때문에 varchar 로 최대 200개 문자열을 받을 수 있도록 했다. Status는 현재 배송상태를 의미하고 대기중, 적재중, 배송중 등이 있을 수 있다. 마지막 location\_id는 place에 대한 세부 정보를 의미하는 것으로 이들 역시 varchar로 최

대 문자열 20개 허용하고 이들 모두 null을 허용해서는 안되는 필수 값들이다.

- Location



location	
location_id:	VARCHAR2(20)
latitude:	FLOAT
longitude:	FLOAT
location_name:	VARCHAR2(200)
emergency_phone_number:	VARCHAR2(20)

예시

('LOC001', 37.1234, -122.5678, 'Main Office', '555-1234')

여기서는 location에 대한 세부정보를 가지는 테이블이다. 해당 위치의 주소는 latitude, longitude로 저장하며 float 데이터 타입으로 저장하고 있다. Location\_name은 주소이기 때문에 얼마나 길지 몰라 200개 문자열을 허용하도록 했다. Emergency\_phone\_number는 해당 location 주소의 임시 연락망으로 문자열 최대 20개로 받도록 했다. 마지막 location\_id는 해당 주소의 primary key로 최대 20개 문자열을 허용하고 이들 모두 null 이 되지 않도록 하였다.

### 3. ODBC implementation within MySQL & explain ODBC C languages codes.

우선 가장 먼저 해야 할 것은 crud 하는 것이다. 명세서에 따르면 txt 파일을 읽어 들여 이를 mysql 에 table을 생성하고 해당 Tuple들을 입력해야 한다. 필자는 두 가지 txt 파일을 사용해서 하나는 테이블 생성 및 Tuple 삽입에 사용했고 나머지는 마지막 프로그램 종료 전에 다시 테이블을 모두 삭제 해줘 다음 실행 시 기존 정보를 가지고 있지 않도록 해줬다.

```

create table recipient(recipient_id varchar(10), name varchar(20) not null, address varchar(200) not null, primary key (recipient_id))
create table customers(customer_id varchar(10), name varchar(20) not null, address varchar(200) not null, primary key (customer_id))
create table recipient_phone_number(phone_number varchar(20), recipient_id varchar(10), primary key(phone_number, recipient_id))
create table services(service_id varchar(10), package_type varchar(10) not null, package_weight float not null, timeliness varchar(20))
create table packages(package_price integer, hazardous_materials varchar(20) not null, international_shipments varchar(20) not null, service_id varchar(10))
create table shipments(shipment_id varchar(10), started_date date not null, expected_date date not null, ended_date date null, customer_id varchar(10))
create table bill_pay(bill_id varchar(10), isrefund smallint not null, date date not null, pay_date date null, customer_id varchar(10))
create table card(credit_card_number varchar(20), customer_id varchar(10) not null, primary key (credit_card_number))
create table account(customer_id varchar(10), account_number varchar(20), monthly_billed_date integer, primary key (customer_id, account_number))
create table location(location_id varchar(20), latitude float not null, longitude float not null, location_name varchar(200) not null, primary key (location_id))
create table tracking(tracking_id varchar(20), shipment_id varchar(10) not null, date timestamp not null, next_place varchar(200) not null, primary key (tracking_id))
INSERT INTO customers (customer_id, name, address) VALUES ('CUST001', 'Do Kim', '123 Seoul Street, Gangnam-gu, Seoul, South Korea')
INSERT INTO customers (customer_id, name, address) VALUES ('CUST002', 'SS Lim', '456 Busan Avenue, Haeundae-gu, Busan, South Korea')
INSERT INTO customers (customer_id, name, address) VALUES ('CUST003', 'Dhara Lee', '789 Incheon Road, Jung-gu, Incheon, South Korea')
INSERT INTO customers (customer_id, name, address) VALUES ('CUST004', 'Sun Yoon', '321 Daegu Lane, Suseong-gu, Daegu, South Korea')
INSERT INTO customers (customer_id, name, address) VALUES ('CUST005', 'James JJ', '654 Gwangju Boulevard, Seo-gu, Gwangju, South Korea')
INSERT INTO recipient (recipient_id, name, address) VALUES ('REC001', 'Jong Sub', '123 Maple Avenue, Gangnam-gu, Seoul, South Korea')
INSERT INTO recipient (recipient_id, name, address) VALUES ('REC002', 'Sea Park', '456 Pine Street, Jongno-gu, Seoul, South Korea')
INSERT INTO recipient (recipient_id, name, address) VALUES ('REC003', 'Gang Kim', '789 Oak Road, Mapo-gu, Seoul, South Korea')
INSERT INTO recipient (recipient_id, name, address) VALUES ('REC004', 'Gang jun Lee', '234 Cherry Lane, Busanjin-gu, Busan, South Korea')
INSERT INTO recipient (recipient_id, name, address) VALUES ('REC005', 'Yoob sang Oh', '567 Elm Court, Yongsan-gu, Seoul, South Korea')
INSERT INTO recipient (recipient_id, name, address) VALUES ('REC006', 'Sung Min', '890 Cedar Drive, Seocho-gu, Seoul, South Korea')
INSERT INTO recipient (recipient_id, name, address) VALUES ('REC007', 'Jin Mun', '321 Willow Avenue, Gwangjin-gu, Seoul, South Korea')
INSERT INTO recipient (recipient_id, name, address) VALUES ('REC008', 'Ji Chae', '654 Birch Lane, Nam-gu, Busan, South Korea')
INSERT INTO recipient (recipient_id, name, address) VALUES ('REC009', 'Ho Moon', '987 Aspen Street, Jung-gu, Busan, South Korea')
INSERT INTO recipient (recipient_id, name, address) VALUES ('REC0010', 'Juna Jin', '432 Chestnut Road, Seodaeun-gu, Seoul, South Korea')
INSERT INTO recipient_phone_number (phone_number, recipient_id) VALUES ('987-654-3210', 'REC001')
INSERT INTO recipient_phone_number (phone_number, recipient_id) VALUES ('876-543-2109', 'REC002')
INSERT INTO recipient_phone_number (phone_number, recipient_id) VALUES ('123-432-1098', 'REC003')
INSERT INTO recipient_phone_number (phone_number, recipient_id) VALUES ('435-654-2343', 'REC004')
INSERT INTO recipient_phone_number (phone_number, recipient_id) VALUES ('456-234-4351', 'REC005')

```

20181256\_crud.txt 파일의 일부를 캡처했다. 각 위에서 설명한 정보들을 토대로 tuple 예시를 만들어 직접 기입하였다.

이렇게 txt파일을 입력하여 fgets 를 사용해 한 줄 씩 읽도록 만들어 졌다.

```

FILE* crud1 = NULL;
fopen_s(&crud1, "20181256_crud.txt", "r");
char read_temp[1000];
int state = 0;
const char* query;
while (!feof(crud1))
{
    query = fgets(read_temp, sizeof(read_temp), crud1);
    state = mysql_query(connection, query);
}
fclose(crud1);

```

로운 가능	20181256.cpp	20181256_crud_d.txt	20181256_crud.txt
1	SET SQL_SAFE_UPDATES = 0		
2	Delete from table tracking		
3	Delete from table location		
4	Delete from table customer_phonenumber		
5	Delete from table account		
6	Delete from table card		
7	Delete from table bill_pay		
8	Delete from table shipments		
9	Delete from table packages		
10	Delete from table services		
11	Delete from table recipient_phone_number		
12	Delete from table customers		
13	Delete from table recipient		
14	DROP TABLE tracking		
15	DROP TABLE location		
16	DROP TABLE customer_phonenumber		
17	DROP TABLE account		
18	DROP TABLE card		
19	DROP TABLE bill_pay		
20	DROP TABLE shipments		
21	DROP TABLE packages		
22	DROP TABLE services		
23	DROP TABLE recipient_phone_number		
24	DROP TABLE customers		
25	DROP TABLE recipient		

20181256\_crud\_d.txt 파일을 캡처했다. 여기서는 tuple들 삭제 및 table 삭제를 해 주는 텍스트 파일이다.

```
FILE* crud2 = NULL;
fopen_s(&crud2, "20181256_crud_d.txt", "r");
char read_temp[1000];
int state = 0;
const char* query;
while (!feof(crud2))
{
    query = fgets(read_temp, sizeof(read_temp), crud2);
    state = mysql_query(connection, query);
}
fclose(crud2);

mysql_close(connection);
```

이 역시 한 줄 씩 읽으면서 이를 mysql\_query 함수를 사용하여 명령어들을 넣어 준다.

여기서는 odbc c언어에 대해 설명을 할 것이다.

우선 명세서의 설명대로 해당 코드들은 input에 0이 들어오기 전까지 계속해서 프로그램을 실행시켜야 한다. 그래서 flag=1를 설정해서 input이 0이면 flag=0 으로 설정하도록 한다. Default는 while(flag)로 계속해서 해당 질문들이 무한 루프를 돌아가도록 해줬다. 그리고 각 type 마다 0~5까지 들어 올 수 있기 때문에 switch 문을 사용해서 각 case 별로 따로 따로 c언어를 작성하였다.

각 type별로 설명하도록 하겠다.

Case 1.

```
----- SELECT QUERY TYPES -----
1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE V
0. QUIT

What Type : 1
---- TYPE I ----
**truck X (hint: 1721) is destroyed in a crash at 2022-04-07**
----- Subtypes in TYPE I -----
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.

We have two truck 1721 and truck 2020
What Detail Type: 1
Truck Num (hint: 1721 ): 1721
**Find all customers who had a package on the truck at the time of the crash**
customer id: CUST003   name: Chang Lee address: 789 Incheon Road, Jung-gu, Incheon, South Korea
customer id: CUST005   name: James JJ  address: 654 Gwangju Boulevard, Seo-gu, Gwangju, South Korea
---- TYPE I ----
```

Case 1번은 트럭 x가 사고가 났을 때 해당 배송을 받지 못한 수령인, 보내지 못한

고객, 그리고 그 전에 가장 최근에 무사히 택배를 받은 사람 3가지 세부 사항으로 나뉜다. 이 역시 Input 0이 들어오기 전까지는 계속해서 무한 루프를 돌 수 있도록 detail\_flag를 뒤서 다시 while(detail\_flag)로 설정했다.

이때 2022-04-07에 1721번 트럭이 사고가 났음을 가정하고 query를 작성하였다.

1-1.

```
Select * from customers where (customer_id) in (select customer_id from shipments where (shipment_id) in (select shipment_id from tracking where next_place='truck ${truck_num}' and date ='2022-04-07 13:00:00'))"
```

이때 strcat\_s를 사용해서 해당 input으로 들어온 truck 번호를 이어 붙여주는 과정이 필요하다. 즉 \${truck\_num} 앞 뒤에 strcat\_s를 사용하면 된다.

이를 mysql\_query에 넣어 mysql\_fetch\_row로 한줄 씩 받아온다.

이 중 고객 id, 이름, 주소 3개의 정보를 보여줬다.

1-2.

```
case 2:
    printf("\nTruck Num (hint: 1721 ): ");
    scanf_s("%s", inputTruck, (int)sizeof(inputTruck));
    printf("\n++Find all recipients who had a package on the truck at the time of the crash++\n");
    strcat_s(query, "select * from recipient where (recipient_id) in (select recipient_id from shipments where (shipment_id) in (select shipment_id from tracking where next_place = 'truck '");
    strcat_s(query, inputTruck);
    strcat_s(query, " and date = '2022-04-07 13:00:00')");
    state = mysql_query(connection, query);
```

여기서는 택배를 받지 못한 수령인에 대한 정보를 출력하는 것이다. 그래서 2022-04-07 13:00 이후의 택배에 대한 shipment\_id를 찾아 shipments table에서 수령인 id를 찾는다. 이후 이를 recipient table에서 다시 원하는 정보인 수령인 id, 이름, 주소를 뽑도록 하였다.

1-3.

```
case 3:
    printf("\nTruck Num (hint: 1721 ): ");
    scanf_s("%s", inputTruck, (int)sizeof(inputTruck));
    printf("\n++Find the last successful delivery by that truck prior to the crash++\n");
    strcat_s(query, "select * from shipments where (shipment_id) in (select max(shipment_id) from tracking where next_place = 'truck '");
    strcat_s(query, inputTruck);
    strcat_s(query, " and status='complete' and date < '2022-04-07 13:00:00' order by date)");
    state = mysql_query(connection, query);
```

Tracking table을 date로 오름차순 정렬 한 뒤 2022-04-07 13:00 이전의 택배 상태가 status='complete'된 shipment\_id를 찾아 max 값 하나를 가지고 이를 shipments table



에서 attribute 들을 select 해서 원하는 정보를 뿌려주도록 하였다.

Case 2 & case 3.

```
char year[10];
char year2[10];
int temp;
scanf("%d", &temp);
memset(query, 0, sizeof(query));
sprintf(year, "%d", temp + 1);
sprintf(year2, "%d", temp);
strcat_s(query, "select customer_id, count(customer_id) as cnt from bill_pay where date<='";
strcat_s(query, year);
strcat_s(query, "-01-01' and date>='");
strcat_s(query, year2);
strcat_s(query, "-01-01' group by customer_id order by cnt desc limit 1");

printf("number year: (%d) \n", temp);
char year3[10];
char year4[10];
int temp2;
scanf("%d", &temp2);
memset(query, 0, sizeof(query));
sprintf(year3, "%d", temp2 + 1);
sprintf(year4, "%d", temp2);
strcat_s(query, "select customer_id, sum(package_price) as price from shipments where shipment_id in (select bill_id from bill_pay where isrefund=0 and started_date<='");
strcat_s(query, year3);
strcat_s(query, "-01-01' and started_date>='");
strcat_s(query, year4);
strcat_s(query, "-01-01' group by customer_id order by price desc limit 1");
state = mysql_query(connection, query);
```

Case 2는 원하는 년도의 택배를 가장 많이 보낸 사람에 대한 정보를 찾는다. Input으로 원하는 년도를 받는데 예를 들어 2022 년이 들어오면 2022.01.01부터 2023.01.01 전까지의 정보를 가지고 있어야 한다. 그래서 해당 년도에 대해 역시 strcat\_s 함수를 사용해 하나의 query로 이어붙일 수 있도록 한다. 이때 customer\_id를 그룹화 하여 이에 대한 횟수를 보고 이에 대해 내림차순을 한 다음 최대 1개만 가져오도록 limit 1을 걸면 최상위 택배를 가장 많이 보낸 사람에 대한 정보를 받아올 수 있다.

Case 3는 원하는 년도에 돈을 가장 많이 쓴 고객에 대한 정보를 찾는다. Case2 와 매우 유사하게 작성할 수 있다. 대신 여기서는 우선 bill\_pay 테이블에서 isrefund가 0인 것을 찾아 환불된 돈은 제외하도록 한다. shipments에 있는 customer\_id 로 group 화 한다음 Package\_price을 모두 더한 sum을 기준으로 내림차순 정렬하고 여기서 1개만 뽑아 최대 로 돈을 많이 쓴 사람에 대한 정보를 받아올 수 있다.

```

What Type : 2
---- TYPE II ----

**Find the customer who has shipped the most packages in the past year**
Which year? ( hint: 2022 ) : 2022
customer id: CUST003 shipped count: 3

----- SELECT QUERY TYPES -----
1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE V
0. QUIT

What Type : 3
---- TYPE III ----

**Find the customer who has spent the most money on shipping in the past year**
Which year? ( hint: 2022 ) : 2022
customer id: CUST003 spent money: 20000won

```

#### Case 4.

```

memset(query, 0, sizeof(query));
strcat_s(query, "select * from shipments where expected_date!=ended_date or ended_date is null order by expected_date");
state = mysql_query(connection, query);

```

여기서는 input이 없다. 그래서 바로 query를 작성하면 되는데 예상 날짜에 택배를 받지 못한 정보들을 찾는다. 이는 shipments 테이블에서 expected\_date와 ended\_date가 다르거나 사고로 인한 택배를 보내지 못한 경우 (null) 을 찾아서 출력하면 된다.

```

What Type : 4
---- TYPE IV ----

**Find the packages that were not delivered within the promised time.**
shipment id: SHIP003 expected date: 2022-02-06 ended_date: 2022-02-07
shipment id: SHIP005 expected date: 2022-03-04 ended_date: 2022-03-05
shipment id: SHIP006 expected date: 2022-03-04 ended_date: 2022-03-05
shipment id: SHIP008 expected date: 2022-04-05 ended_date: 2022-04-07
shipment id: SHIP010 expected date: 2022-04-07 ended_date: (null)(Truck was crashed)
shipment id: SHIP009 expected date: 2022-04-07 ended_date: (null)(Truck was crashed)
shipment id: SHIP013 expected date: 2023-05-05 ended_date: 2023-05-06

----- SELECT QUERY TYPES -----
1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE V
0. QUIT

```

#### Case 5.

```

sprintf(year5, "%d-", temp5);
sprintf(month1, "%d-", temp6);
sprintf(year6, "%d-", temp7);
sprintf(month2, "%d-01-", temp8);

printf("\n== Bill List at year: %d month: %d ==\n", temp5, temp6);
memset(query, 0, sizeof(query));
strcat_s(query, "select c.customer_id, c.name, c.address, s.package_price, pay_date, service_id, b.date from bill_pay as b inner join shipments as s inner join customers as c inner join packa");
strcat_s(query, year5);
strcat_s(query, month1);
strcat_s(query, "-01" and s.started_date<");
strcat_s(query, year6);
strcat_s(query, month2);

state = mysql_query(connection, query);

```

여기서는 년도, 달을 input으로 받아 해당 달에 대해 bill을 생성해서 보여주는 것이다. 위의 캡처본에 잘렸는데 달마다 일수가 다르기 때문에 만약 2023년 1월이 들어온다면

2023.01.01 부터 2023.02.01 전까지의 정보를 가져오려고 했다. 이때 주의해야 할 점은 만약 12월 이 input으로 들어온다면 년도도 1개 올라가고 다음달은 1월로 새로 설정을 해야 한다. 이 과정을 거친 후 shipment table을 기준으로 원하는 년도 달에 대해 shipemnt\_id 들을 가져온 다음, join을 사용해서 뽑고자하는 정보들을 모두 뽑아 왔다. 우선 해당 bill이 생성된 날짜를 bill\_pay 테이블에서 가져오고 customer id, customer name, customer address 는 customer 테이블에서 가져온다. 그리고 bill\_pay table에 있는 pay\_date 를 통해 null이면 돈을 안냈기 때문에 돈을 내라는 “You have to pay!” 라는 문구를 보내준다. 각 service\_id 역시 package table에서 뽑아와서 어떤 서비스를 이용했는지에 대해 나타나게 된다. 마지막으로 해당 bill에 대한 돈 역시 shipments 테이블의 package\_price에서 가져온다. 이렇게 원하는 년도, 달에 대해 각 택배 별로 따로 bill이 출력된다.

```
Input Year, Month ( hint: 2022 4 ):2022 4
** Bill List at year: 2022 month: 4 **
Date: 2022-04-04 Customer id: CUST005 Customer name: James JJ Customer address: 654 Gwangju Boulevard, Seo-gu, Gwangju, South Korea
Service_id: SRV005 Each Price: 8000 won(You have to pay!)
Date: 2022-04-01 Customer id: CUST004 Customer name: Sun Yoon Customer address: 321 Daegu Lane, Suseong-gu, Daegu, South Korea
Service_id: SRV001 Each Price: 3000 won(Paid)
Date: 2022-04-03 Customer id: CUST003 Customer name: Chang Lee Customer address: 789 Incheon Road, Jung-gu, Incheon, South Korea
Service_id: SRV001 Each Price: 3000 won(Paid)
```