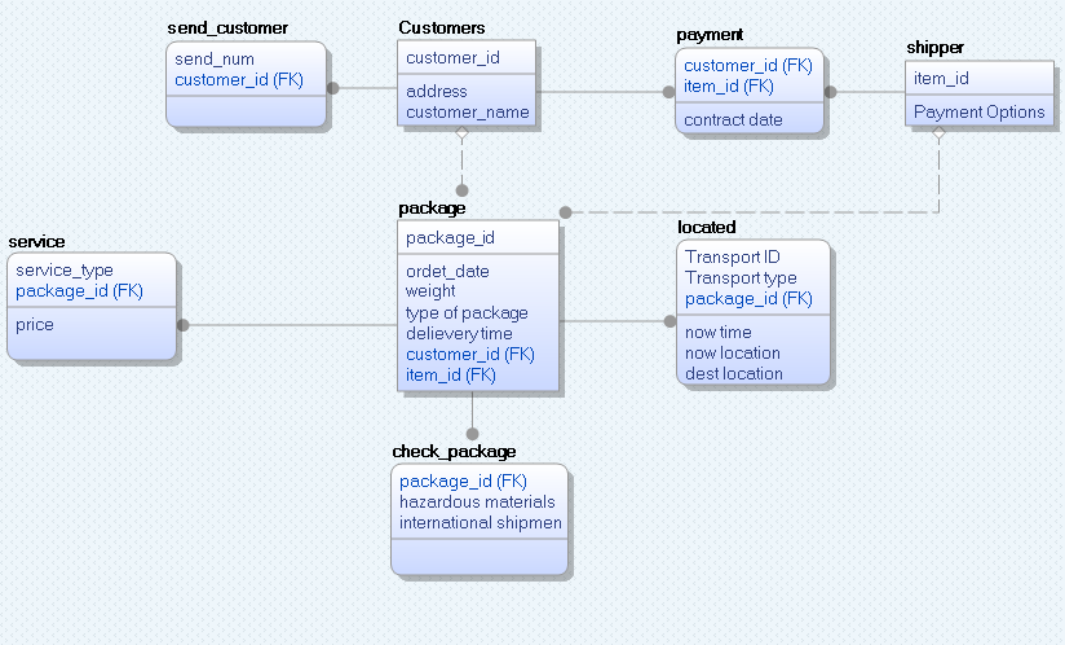
[CSE4110]데이터베이스 시스템 Project 2

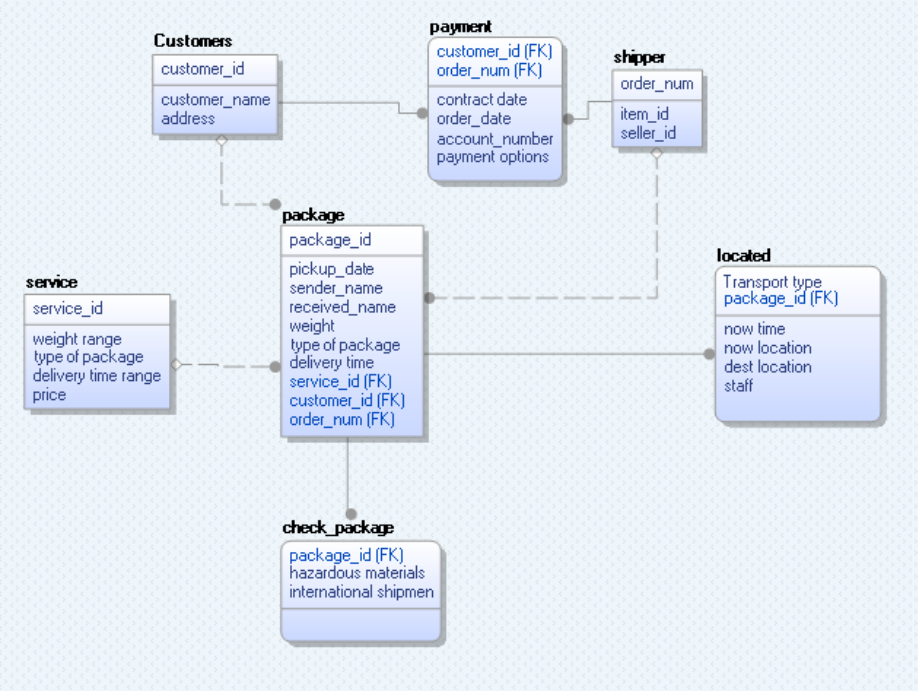
20200118 조창희

1. Original Logical Schema



원래 논리적 스키마를 위와 같이 작성하였으나, 평가를 받으면서, 그리고 프로젝트 2를 수행하면서 기존 논리적 스키마의 한계/불필요한 사항들을 발견하고 수정하였습니다.

2. Revised Logical Schema



2-1 : send_customer entity 삭제 : 쿼리문(count)으로도 충분히 customer이 발송한 횟수를 구할 수 있으므로, 불필요한 entity라고 느껴져 삭제하였습니다.

2-2 : payment entity 변경 : payment option을 구분자로 하여, 만약 payment option이 계약이라면 contract date에 계약 날짜가 표시됩니다. 또한 상품을 주문한 날짜인 order date와 account number 속성이 추가되었습니다. 고객이 청구하기 위해서는 계좌 정보가 필요할 것이라는 생각이 들었습니다.

2-4: shipper entity 정보 추가 : 저는 shipper을 쿠팡, 위메프와 같은 실제 인터넷 사이트라고 가정하였습니다. 해당 사이트에서 물건을 사기 위해서는 판매자 정보, 물건 정보가 필수적으로 필요하다고 생각하여 추가하였습니다. 또한 구매자가 주문한 날짜인 order date 또한 추가하였습니다.

2-5: package entity 정보 추가/변경 : 기존 order date를 삭제하고, customer이 실제로 배송을 보낸 날짜 혹은 shipper이 배송을 보낸 날짜인 pickup_date가 추가되었습니다. 그리고 보내는 사람과 받는 사람의 정보인 sender name과 received name 정보를 추가하였습니다.

2-6: located entity 변경 : package_id(FK)를 제거하였습니다. 배송 직원인 staff의 정보가 추가되었습니다.

2-7: service entity 변경 : 기존에는 service의 PK에 package id를 포함하였으나, service의 종류는 package의 종류보다 적으므로 무의미한 정보의 중복이 발생할 것이라는 생각이 들었습니다. 따라서 service_id만 PK에 포함한 뒤, package의 FK로 추가하는 방식으로 하여 정보의 중복을 최소화하였습니다.

(참고) **package entity에 대한 설명** : 저는 고객이 배송을 보낼 때도 있으며, 고객과 거래를 거친 인터넷 쇼핑몰이 배송을 보낼 때도 있다는 가정 하에 설계를 진행하였습니다. 따라서 null을 최소화하며 두 배송을 구분할 수 있는 방법에 대해 여러 고민을 하였습니다. 처음에는 두 entity를 추가하여 각각의 package를 관리하고자 하였으나, 속성의 중복이 두배나 발생한다는 사실 때문에 널값을 허용하더라도 하나의 package로 관리하도록 결정하였습니다

또한, 이후 쿼리문을 처리하는 데에 있어, '보내는 사람'의 범위를 어떻게 할 것인가 고민을 하였습니다. 처음에는 개인적으로 보내면 customer로, 쿠팡과 같은 shipper 업체를 통해 보내면 쿠팡이 보낸 것으로 정의하려고 하였습니다. 하지만 다음과 같은 두 가지 문제가 있었습니다.

1. shipper을 통해서 보내도 customer이 직접 배송비를 내고 주문한다는 점에서 customer이 보낸다고 보는 것이 더 합리적입니다.
2. 또한, 만약 보내는 것을 쿠팡과 같은 shipped로 한다면 '가장 많이 보낸 고객'이라는 질문의 답이 무조건 쿠팡이 될 것 같아, 고객 별 분리가 필요하다는 생각이 들었습니다.

따라서 해당 entity에서는 shipper이 발송하는 경우에도 customer이 보낸다고 보았고, 다만 개인적인 배송과 비교했을 때 다음 두 가지 구별점을 두었습니다.

1. 개인이 발송하는 경우 order_num이 NULL이 되도록 하였습니다.
2. Sender_name에서 shipper을 통해 보내는 경우, 고객명(shipper) 과 같은 형식으로 기록하도록 하였습니다.

따라서 자동으로 이 두 종류의 발송을 구분할 수 있습니다.

3. BCNF Decomposition

2번에서 수정한 logic schema를 바탕으로 BCNF(Boyce-Codd Normal Form) 분해를 적용할 것입니다. 알파와 베타가 모두 R의 부분집합이라고 가정할 때, 다음 두 조건 중 적어도 하나의 조건을 만족하면 BCNF라고 할 수 있습니다.

$\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
α is a superkey for R

따라서 저는 모든 FD를 구하고, (1) trivial한지, (2) superkey인지를 따진 다음 둘 다 해당하지 않은 경우 Decomposition을 수행하는 방식으로 진행하도록 하겠습니다.

3-1: service

Service_id -> weight range, type of package, delivery time range, price

Weight range, type of package, delivery time range -> price

첫번째 dependency의 경우, service_id가 PK이므로 (2) 조건을 만족합니다. Weight range, type of package, delivery time range 역시, 이 세 조건이 합쳐지면 나머지 속성을 모두 구분할 수 있으므로 super key에 해당합니다. 따라서 해당 entity는 BCNF가 성립합니다

3-2: Customers

Id 외에 FD가 나타나지 않습니다. 따라서 BCNF를 만족합니다.

3-3: check_package

PK로만 구성되어있으므로 역시 FD가 나타나지 않습니다.

3-4:. Located

PK 외의 FD는 나타나지 않습니다.

3-5: shipper

order_num -> item_id, seller_id

item_id -> seller_id

item_id는 PK는 아니지만 super key에 해당합니다. 해당 entity의 모든 속성을 구분지을 수 있기 때문입니다. 따라서 BCNF에 속합니다.

3-6: payment

PK외의 FD가 드러나지 않습니다. (payment option이 계약이라면 contract_date가 널이 아니라 특정 값인 것까지는 알 수 있겠지만, 정확한 날짜는 모르므로 dependency가 성립한다고 보기 어렵습니다.) 따라서 BCNF에 속합니다.

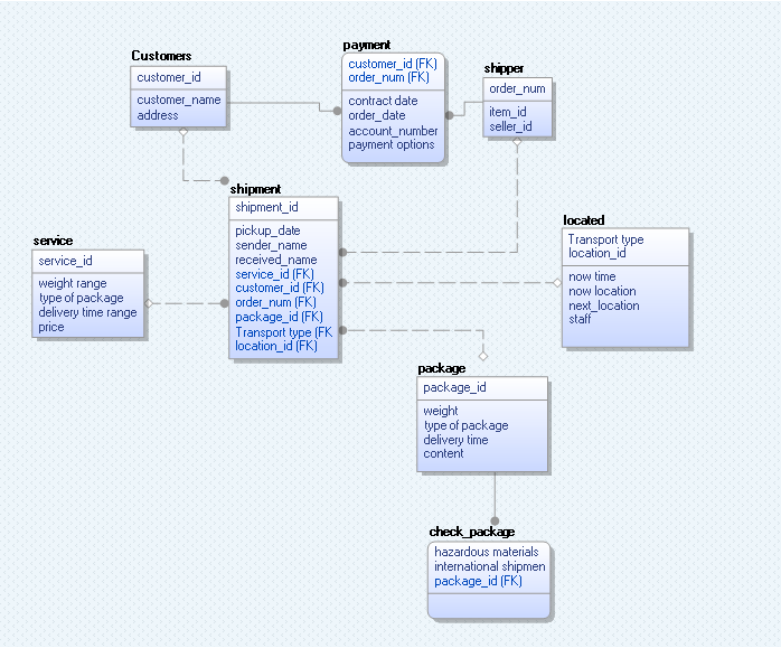
3-7: package

package_id -> pickup_date, sender_name, received_name, weight, type of package, delivery time, service_id, customer_id, order_num

weight, type of package, delivery time -> service_id

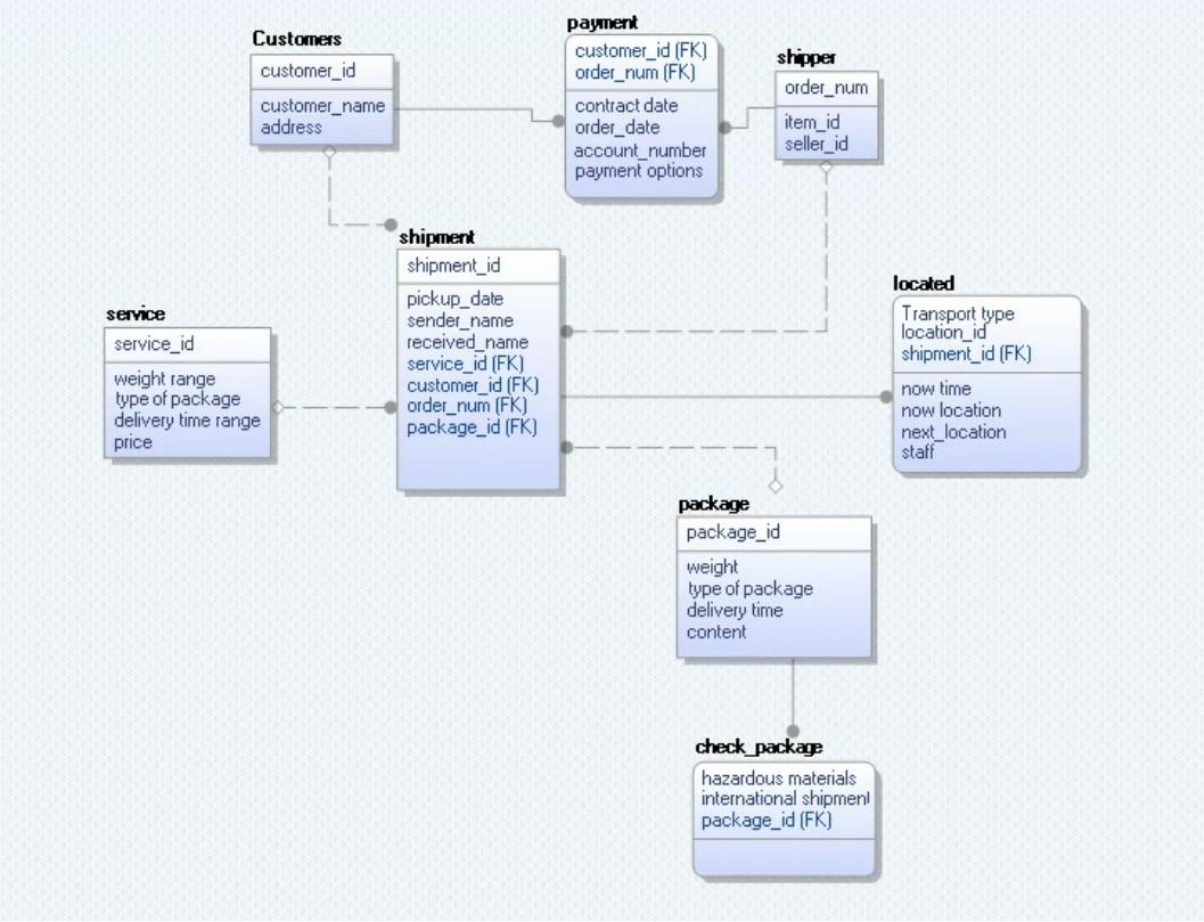
weight, type of package, delivery time의 조합은 해당 entity의 super key라고 보

기는 어렵습니다. 왜냐하면 다른 것들은 유일하게 결정시키지 못하기 때문입니다. 그리고 trivial하지도 않습니다. 다른 집합의 부분집합이라고 보기 어렵습니다. 따라서 BCNF를 충족하지 않습니다. 따라서 decomposition을 시켜주어야 합니다.



위와 같이 변경해주었습니다. package entity를 따로 빼서 저장하였고, content 속성을 추가해줬습니다. 그리고 구 package entity였던 shipment entity에 FK로 연결시켜주었습니다. 이렇게 하면 package_id, service_id 사이에 FD가 생기지 않습니다. 또한 check_package를 새롭게 생성한 package entity에 연결시켜주었습니다. 이 외에 더 이상 BCNF를 위반하는 entity가 없으므로, decomposition을 중단합니다.

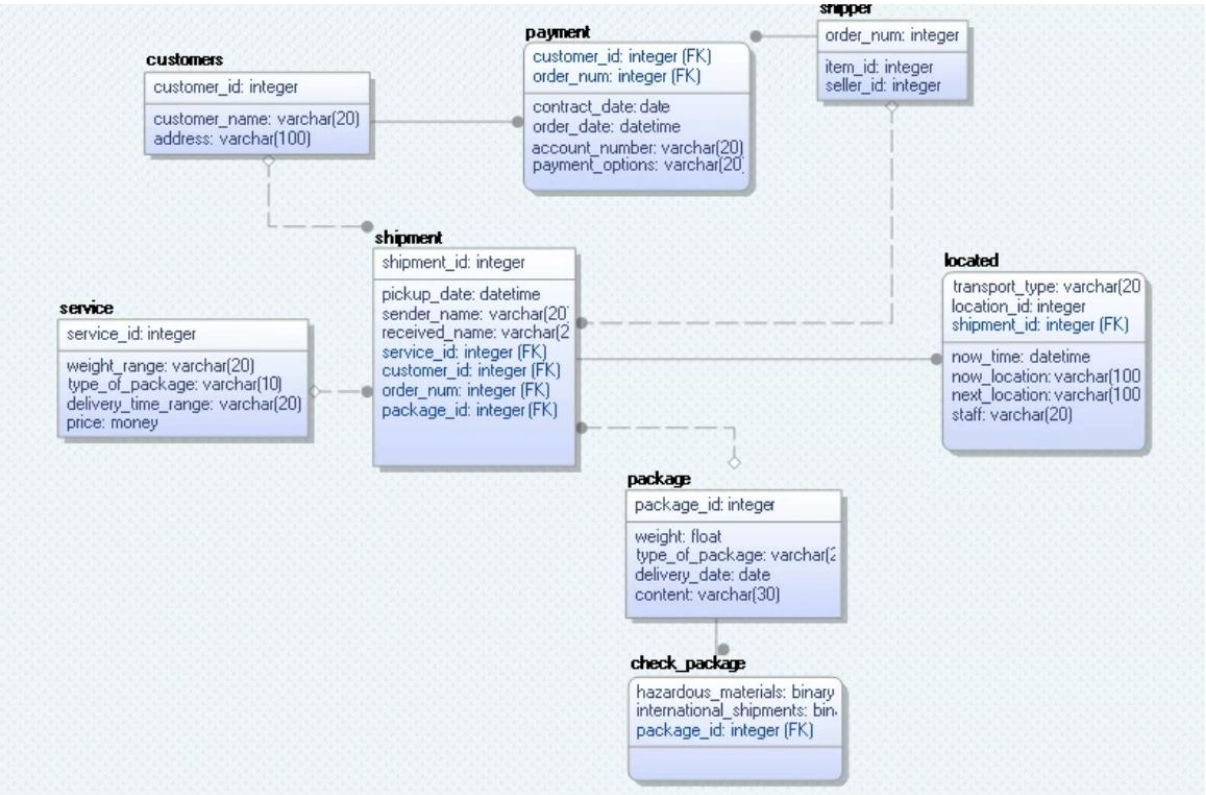
4. 3차 수정



이후 쿼리문을 작성하면서 3차 수정을 거쳤습니다. location의 attribute도 추가로 수정을 해주었습니다. 여기서 location_id는 마을 버스에 있는 숫자와 같은 의미입니다. 특정 지역 위치를 기반으로 하고 있으나, 현재 location이 아예 변경되지는 않습니다. 따라서 수화물들 간 구분이 있기 위해서는 운송 타입 + 지역 위치 + shipment id(운송번호) 모두가 있어야 식별이 가능합니다. 따라서 위와 같이 수정을 거쳤습니다. 위 사진이 최종 logical schema입니다.

5. Physical Schema Diagram

Erwin 가이드 문서를 참고하여, 의도한 테이블에 맞게 domain과 NULL 여부 등을 수정하는 과정을 거쳤습니다. 기본적으로는 Not NULL로 하되, NULL이어도 상관없는 경우라고 판단되면 허용하였습니다. 총 8개 table에 대한 변환이 이루어졌으며, 다음은 변환 결과입니다.



해당 physical schema에 대하여 아래와 같이 표로 정리하였으며, 설명이 필요한 경우에는 아래 설명을 추가하였습니다.

table	attribute	key	domain	Null 가능 여부
customers	customer_id	PK	integer	Not NULL
	customer_name		Varchar(20)	Not NULL
	address		Varchar(100)	Not NULL
payment	customer_id	PK, FK	integer	Not NULL
	order_num	PK, FK	integer	Not NULL
	contract_date		date	NULL
	order_date		datetime	Not NULL
	Account_number		Varchar(20)	Not NULL
	Payment_options		Varcjar(20)	Not NULL
shipper	Order_num	PK	Integer	Not NULL
	Item_id		Integer	Not NULL
	Seller_id		Integer	Not NULL
	service_id	PK	Integer	Not NULL

service	Weight_range		Varchar(20)	Not NULL
	Type_of_package		Varchar(10)	Not NULL
	Delivery_time_range		Varchar(20)	Not NULL
	price		money	Not NULL
Check_package	package_id	PK, FK	Integer	Not NULL
	Hazardous_materials	PK	Binary	Not NULL
	International_shipments	PK	binary	Not NULL
package	package_id	PK	Integer	Not NULL
	Weight		float	Not NULL
	Type_of_package		varchar(20)	Not NULL
	Delivery_date		date	Not NULL
	Content		Varchar(30)	NULL
shipment	Shipment_id	PK	integer	Not NULL
	service_id	FK	integer	Not NULL
	Customer_id	FK	integer	Not NULL
	Order_num	FK	integer	NULL
	Package_id	FK	integer	Not NULL
	Pickup_date		datetime	Not NULL
	sender_name		Varchar(20)	Not NULL
	received_name		Varchar(20)	Not NULL
located	transport type	PK	Varchar(20)	Not NULL
	Shipment_id	FK, PK	integer	Not NULL
	Location_id	PK	integer	Not NULL
	now_time		daytime	Not NULL
	now location		varchar(100)	Not NULL
	next_location		varchar(100)	NULL
	staff		varchar(20)	Not NULL

5-1. customers

id는 숫자로 표시할 계획이므로(기본적으로 4자리), customer_id 또한 숫자 자료형을 도메인으로 갖습니다. customer name과 address는 고정된 값이 아니므로 varchar로 선언하였으며, 모두 넉넉한 글자수로 각각 20, 100을 할당하였습니다. 해당 테이블의 attribute는 모두 이용에 있어서 필수적으로 필요한 정보이기 때문에 널값을 허용하지

않습니다.

5-2. shipper

order num은 총 10글자 (customer id 4자리 + item id 4자리 + 주문 개수별)로 만들어서 넣을 계획입니다. 따라서 integer로 선언합니다. Item id와 seller id모두 4자리이다. 3정보 모두 필수 정보이므로 Not NULL로, 널값을 허용하지 않습니다.

5-3. Payment

Customer과 shipper 사이 지불 정보에 대한 테이블이다. 두 테이블의 PK의 조합을 해당 테이블의 PK로 합니다. 그리고 contract date는 날짜까지는 알 필요가 있으나 시간까지는 불필요한 정보라고 생각하여 date로 정하였고, payment option에 따라 필요하지 않을 수도 있어, 널값을 허용합니다. Order date는 실제로 상품을 주문한 시각으로, 정확한 시간 정보가 필요하기 때문에 datetime으로 설정합니다. Account_number은 보통 10~15자리이며, 은행의 종류도 포함하고 있어야하므로 varchar(20)으로 설정하였고, payment option은 prepaid, credit card, contract 세가지 옵션에 따라 나뉩니다. (여기서 계좌는 shipped의 계좌이며, 고객이 청구해야 합니다. 어떠한 payment option이든 shipped의 계좌를 통해 이루어진다고 가정합니다.)

5-4. Service

Weight_range는 예로 0~10, 100 ~ 500, 500~1000, 1000~1500과 같이 표시하며, 단위는 g(그람)입니다. Type of package는 flat envelope, small bax 등이 있습니다. Time range은 one day, two days ..로 표시하며, 이에 따라 가격이 달라집니다. 모든 속성은 널을 허용하지 않습니다.

5-5. Check_package

Package를 감시하고 있으며, 역한 개체이므로 package_id, international shipments, hazardous materials의 조합이 PK입니다. 또한 hazardous, international은 binary domain을 적용하여 0과 1로 구분하고자 하였습니다.

5-6. Package

Weight, type of package, delivery time 등 해당 패키지에 대한 정보가 담겨 있습니다. 여기서 Delivery date은 예상 도착 날짜를 의미합니다. Content의 경우 안의 물품에 대한 정보인데, 필수 정보는 아니기 때문에 널값을 허용으로 하였습니다.

5-7. Shipment

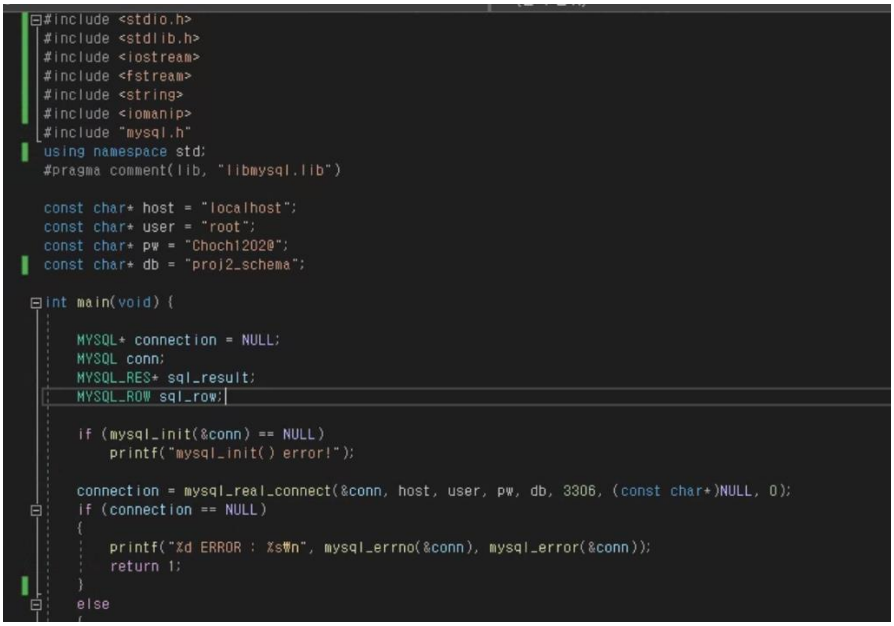
배송과 관련된 전반적인 정보를 저장하는 table입니다. 보내는 사람과 받는 사람의 정보를 받고, 실제 택배가 수거되는 시간 정보가 저장됩니다. 여기서 order_num은 NULL이 허용되는데, 까닭은 인터넷 쇼핑물 회사가 보내는지 개인이 직접 보내는지 여부에 따라 NULL값이 각각 생기기 때문입니다.

5-8. Located

Next location의 경우 도착하면 그 다음이 있을 수 없으니 NULL값을 허용하였습니다. 배송의 완료 여부는 next location의 널 여부를 통해 알 수 있습니다. 현재 운송수단(혹은 창고)에 대한 고유 id, 시간 정보, 담당자 정보등을 담았습니다. Staff는 배송원의 이름 정보를 담았습니다.

6. ODBC implementation within MySQL

명세서대로 dll이 아닌, txt파일을 통한 ODBC가 가능하도록 하였습니다. 우선, MySQL workbench에서 db를 생성하고, MySQL 서버와 비쥬얼 스튜디오를 연결하기 위해 host, db명, pw등을 설정하고 mysql_real_connect()를 호출시켰습니다.



```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
#include "mysql.h"
using namespace std;
#pragma comment(lib, "libmysql.lib")

const char* host = "localhost";
const char* user = "root";
const char* pw = "Choch12020";
const char* db = "proj2_schema";

int main(void) {

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

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

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

파일을 한번 읽을 때 끝까지 읽도록 설정해두었기 때문에 정보 생성하는 txt파일과 정보를 삭제하는 txt파일을 분리하였습니다. 20200118_1.txt는 table create, insert에 대한 정보를 담았고, 20200118_2.txt는 table drop에 대한 정보를 담았습니다. txt파일의 경우 physical schema와 예시문을 참조하여 직접 메모장에 적었습니다.

```
//table create, insert

ifstream file("20200118_1.txt");
string line;
if (file.is_open())
{
    while (getline(file, line))
    {
        int check = 0;
        check = mysql_query(connection, line.c_str());
        if (check == 1)
            printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
        else if (check == 0)
            cout << "success" << endl;

    }
    file.close();
}
else
{
    cout << "File open error";
    return 1;
}
```

```
//drop table
ifstream cfile("20200118_2.txt");
string cline;
if (cfile.is_open())
{
    while (getline(cfile, cline))
        mysql_query(connection, cline.c_str());
}
else
{
    cout << "File open error";
    return 1;
}
cfile.close();
mysql_close(connection);
```

<위: 파일 정보를 읽는 코드>

<txt파일 일부>

20200118_1.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
create table customers(customer_id integer primary key, customer_name varchar(20), address varchar(100));
create table shipper(order_num integer primary key, item_id integer,seller_id integer);
create table payment(customer_id integer, foreign key(customer_id) references customers(customer_id),order_num integer, foreign key(order_num) references shipper(order_n
create table service(service_id integer primary key, weight_range varchar(20), type_of_package varchar(20), delivery_time_range varchar(20), price integer);
create table package(package_id integer primary key,weight float, type_of_package varchar(20),delivery_time date,content varchar(30));
create table check_package(hazardous_materials binary, international_shipments binary, package_id integer, foreign key(package_id) references package(package_id), primary k
create table shipment(shipment_id integer primary key,pickup_date datetime,sender_name varchar(20),received_name varchar(20),service_id integer,foreign key(service_id) refe
create table located(transport_type varchar(20),location_id integer, shipment_id integer, foreign key(shipment_id) references shipment(shipment_id), primary key(transport_typ

insert into customers values (0001, "changhee", "Gangdong-gu, Seoul, Republic of Korea");
insert into customers values (0002, "coco", "Tongyeong-si, Gyeongsangnam-do, Republic of Korea");
insert into customers values (0003, "sehwa", "240, Olympic-ro, Songpa-gu, Seoul, Republic of Korea");
insert into customers values (0004, "sogang", "118, Seogang-ro, Mapo-gu, Seoul, Republic of Korea");
insert into customers values (0005, "star", "1955, Goyang-daero, Deogyang-gu, Goyang-si, Gyeonggi-do, Republic of Korea");
insert into customers values (0006, "jane", "40, Namdong-gil, Gwangsan-gu, Gwangju, Republic of Korea");
insert into customers values (0007, "naver", "95, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, Republic of Korea");
insert into customers values (0008, "minho", "Songpa-gu, Seoul, Republic of Korea");
insert into customers values (0009, "42", "Gangnam-gu, Seoul, Republic of Korea");
insert into customers values (0010, "yelin", "Gangbuk-gu, Seoul, Republic of Korea");
insert into customers values (0011, "sohyun", "Bundang-gu, Seongnam-si, Gyeonggi-do, Republic of Korea");
insert into customers values (0012, "google", "345 Spear Street San Francisco, CA 94105");
insert into customers values (0013, "umea", "UNIVERSITETSTORGET 4, 901 87 Umea, Sweden");
insert into customers values (0014, "Paul", "2000 Las Vegas Blvd S, Las Vegas, NV 89104 United States");
```

20200118_2.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
drop table;
drop table shipment;
drop table check_package;
drop table package;
drop table located;
drop table service;
drop table payment;
drop table shipper;
drop table customers;
```

7. query implementation

기본적으로 명세서와 최대한 비슷하게 구현하고자 하였습니다. Txt파일을 모두

읽어드린 다음, while(1)로 0을 입력할 때까지 무한 반복을 합니다. 1이 입력될 경우 세가지의 서브타입 창으로 넘어갑니다.

```
while (1)
{
    cout << "--------------- SELECT QUERY TYPES ---------------" << endl;
    cout << endl;
    cout << "1. TYPE I" << endl;
    cout << "2. TYPE II" << endl;
    cout << "3. TYPE III" << endl;
    cout << "4. TYPE IV" << endl;
    cout << "5. TYPE  V" << endl;
    cout << "0. QUIT" << endl;
    cin >> input;
    if (input == 0) //quit
    {
        cout << "Good Bye :)" << endl;
        break;
    }
    else if (input == 1) //execute sub type
    {
        while (1)
        {
            string truck;
            cout << "--------------- Subtypes in TYPE I ---------------" << endl;
            cout << "1. TYPE I-1." << endl;
            cout << "2. TYPE I-2." << endl;
            cout << "3. TYPE I-3." << endl;
            cout << "0. QUIT" << endl;
            cin >> input;
            if (input == 0)
            {
                cout << "Go back to the query select menu" << endl;
                break;
```

1. 입력으로 주어진 번호의 트럭이 충돌되었다고 가정하고, (1-1) 충돌된 트럭 안에 있던 물품의 보내는 사람 목록 출력, (1-2) 충돌된 트럭 안에 있던 물품의 받는 사람 목록 출력, (1-3) 충돌되기 전에 마지막으로 성공한 배달을 출력하는 방식으로 구현하였습니다.

   저는 txt파일로 7777 plane를 거친 수화물 1개, 1111 truck을 거친 수화물 4개, 총 5가지에 대한 attribute를 추가하였습니다.

   4개 중 2개의 수화물은 이미 무사히 배송이 되었고, 나머지 두개는 트럭이 운송하던 중에 충돌이 되었습니다. 이를 txt파일로 나타내면 아래와 같습니다.

```
nsert into package values (010200, 400.0, "small box","2023-06-10", "books");
nsert into package values (030400, 200.0, "small box","2023-06-07", "puzzle");
nsert into package values (030401, 200.0, "small box","2022-06-07", "puzzle");
nsert into package values (050000, 550.0, "big box","2023-06-10", "dolls");
nsert into package values (011200, 300.0, "small box","2023-06-06", "ipad");

nsert into check_package values ('0', '0', 010200);
nsert into check_package values ('0', '0', 030400);
nsert into check_package values ('0', '0', 030401);
nsert into check_package values ('0', '0', 050000);
nsert into check_package values ('0', '1', 011200);

nsert into shipment values (77010200, "2023-06-06", "changhee", "coco", 2002, 0001, NULL, 010200);
nsert into shipment values (77030401, "2022-06-05", "sewha", "sogang", 2002, 0003, NULL, 030401);
nsert into shipment values (77030400, "2023-06-05", "sewha", "sogang", 2002, 0003, NULL, 030400);
nsert into shipment values (77050000, "2023-06-06", "Coupang", "star", 3002, NULL, 0005001401, 050000);
nsert into shipment values (77011200, "2023-06-04", "changhee", "google", 2002, 0001, NULL, 011200);

nsert into located values ("truck", 1111, 77030401, "2022-06-07", "sogang, 118, Seogang-ro, Mapo-gu, Seoul, Republic of Korea", NULL, "lyuminho");
nsert into located values ("truck", 1111, 77010200, "2023-06-08", "Gangdong-gu, Seoul, Republic of Korea", "Songpa-gu, Seoul, Republic of Korea", "lyuminho");
nsert into located values ("truck", 1111, 77030400, "2023-06-07", "sogang, 118, Seogang-ro, Mapo-gu, Seoul, Republic of Korea", NULL, "lyuminho");
nsert into located values ("truck", 1111, 77050000,  "2023-06-08", "Gangdong-gu, Seoul, Republic of Korea", "Songpa-gu, Seoul, Republic of Korea", "lyuminho");
nsert into located values ("plane", 7777, 77011200, "2023-06-08", "Gangseo-gu, Seoul, Republic of Korea", "Songpa-gu, Seoul, Republic of Korea", "lyuminho");
```

1-1. 먼저 사고가 날 트럭을 입력으로 받고, 다음과 같은 쿼리문을 작성하였습니다.

select shipment_id, sender_name

from shipment natural join located

where location_id = truck (트럭 번호) and transport_type LIKE 'truck' and next_location is not NULL

동명이인이 있을 것을 간주하여, 저는 사고가 난 수화물을 보낸 사람의 이름과 운송번호를 함께 출력하도록 하였습니다. 보낸 사람의 이름을 알기 위해 shipment table과 located table을 natural join하였다. 그리고 입력으로 받은 번호, truck 운송수단, next_location이 NULL이 아닌 정보를 출력하도록 하였습니다. NULL 여부를 판단한 이유는, 저는 배송 완료가 되었을 경우 next_location을 NULL로 표시하였기 때문입니다. 아직 운송중인 상태의 수화물들만 보아야하기 때문에 위 조건을 추가하였습니다.

그리고 입력받은 번호의 트럭이 존재하지 않거나, 이미 모든 수화물에 대해 배송이 완료된 트럭이라면 "Trucks of that number do not exist이 출력되도록 하였습니다. 현재는 데이터 수가 줄어서 1111 외에는 모두 위 출력이 나오겠지만, 추후 데이터가 추가된다면 다양한 트럭 번호에 대한 추적이 가능할 것입니다.

```
        cout << "go back to the query select menu" << endl;
        break;
    }
    else if (input == 1)
    {
        cout << "Assume truck X is destroyed in a crash." << endl;
        cout << "Find all customers who had a package on the truck at the time of the crash." << endl;
        cout << "Which Truck? : ";
        cin >> truck;
        string query = "select shipment_id, sender_name from shipment natural join located where location_id = " + truck + " and transport_type LIKE 'truck' and next_location is not NULL";
        tmpstate = mysql_query(connection, query.c_str());
        if (tmpstate == 0)
        {
            sql_result = mysql_store_result(connection);
            int cnt = 0;
            while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
            {
                cout << "shipment id : " << sql_row[0] << " sender name : " << sql_row[1] << endl;
                cnt++;
            }
            mysql_free_result(sql_result);
            if (cnt == 0)
                cout << "Trucks of that number do not exist" << endl;
        }
```

<구현 코드>

```
--------------- Subtypes in TYPE I ---------------
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.
0. QUIT
1
Assume truck X is destroyed in a crash.
Find all customers who had a package on the truck at the time of the crash.
Which Truck? : 1111
shipment id : 77010200 sender name : changhee
shipment id : 77050000 sender name : star(Coupang)
--------------- Subtypes in TYPE I ---------------
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.
0. QUIT

1
Assume truck X is destroyed in a crash.
Find all customers who had a package on the truck at the time of the crash.
Which Truck? : 1234
Trucks of that number do not exist
--------------- Subtypes in TYPE I ---------------
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.
0. QUIT
```

<구현 결과>

1-2. 마찬가지로 사고가 날 트럭을 입력으로 받고, 다음과 같은 쿼리문을 작성하였습니다.

select shipment_id, received_name

from shipment natural join located

where location_id = truck and transport_type LIKE 'truck' and next_location is not NULL

reveived_name을 select한다는 점을 제외하면 1-1번과 같은 쿼리문입니다.

```
else if (input == 2)
{
    cout << "Assume truck X is destroyed in a crash." << endl;
    cout << " Find all recipients who had a package on that truck at the time of the crash" << endl;
    cout << "Which Truck? : ";
    cin >> truck;
    string query = "select shipment_id, received_name from shipment natural join located where location_id = " + truck + " and transport_type LIKE 'truck' and next_location is not NULL";
    tmpstate = mysql_query(connection, query.c_str());
    if (tmpstate == 0)
    {
        sql_result = mysql_store_result(connection);
        int cnt = 0;
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            cout << "shipment id : " << sql_row[0] << " received name : " << sql_row[1] << endl;
            cnt++;
        }
        mysql_free_result(sql_result);
        if (cnt == 0)
            cout << "Trucks of that number do not exist" << endl;
    }
```

<구현 코드>

```
--------------- Subtypes in TYPE I ---------------
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.
0. QUIT
2
Assume truck X is destroyed in a crash.
 Find all recipients who had a package on that truck at the time of the crash
Which Truck? : 1111
shipment id : 77010200 received name : coco
shipment id : 77050000 received name : star
--------------- Subtypes in TYPE I ---------------
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.
0. QUIT
2
Assume truck X is destroyed in a crash.
 Find all recipients who had a package on that truck at the time of the crash
Which Truck? : 1010
Trucks of that number do not exist
--------------- Subtypes in TYPE I ---------------
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.
0. QUIT
```

<구현 결과>

1-3. 마찬가지로 사고가 날 트럭을 입력으로 받고, 다음과 같은 쿼리문을 작성하였습니다.

select shipment_id, received_name, now_time

from shipment natural join located

where location_id = truck and transport_type LIKE 'truck' and next_location is NULL

order by now_time desc limit 1

여기서 now_time은 배송 당시에 현재 시각이므로, 배송 완료 시각이라고

할 수 있다. 배송이 이미 완료된 수화물들을 출력해야하기 때문에, next_location이 NULL인 것에 대해서만 select를 수행합니다.

또한, last successful delivery이므로, 이중에서 가장 최근 날짜의 배송을 출력해야합니다. 따라서 시간을 기준으로 내림차순으로 정렬한 뒤, 그 중 하나의 배송건만 출력하도록 합니다. 만약 limit 1을 하지 않는다면, 제 데이터 상에 77030401와 77030400 두 배송건이 모두 출력될 것입니다.

2. 연도를 입력으로 받고, 해당 연도에 가장 배송을 많이 보낸 고객을 출력해야 합니다. Page 3에서 명시한 것처럼, 저는 업체를 통해 배송을 보낸 경우도 고객이 배송을 보냈다는 가정하에 두었습니다. 또한 쿼리가 Find the customer 이라고 되어있고, 특별한 제약사항이 없기 때문에, 가장 많이 배송을 보낸 고객은 한명이라는 가정 하에 쿼리문을 작성하였습니다. 작성한 쿼리문은 다음과 같습니다.

   SELECT customer_id, COUNT(shipment_id) AS shipment_count, customer_name

   FROM shipment natural join customers

   WHERE YEAR(pickup_date) = year

   GROUP BY customer_id

   ORDER BY shipment_count DESC

   LIMIT 1

Customer_id를 기준으로 그룹화를 하고, 각 customer마다 count(shipment_id)를 통해 배송을 보낸 횟수를 계산합니다. 그리고 횟수를 기준으로 내림차순을 하여 가장 많이 보낸 고객을 최상단에 두고, limit 1을 통해 이 경우에 대해서만 출력을 수행합니다. 마찬가지로 해당 연도에 데이터가 없는 경우에 대해서도 예외처리를 해주었습니다. 제가 작성한 CRUD.txt는 다음과 같습니다. 데이터에 따르면 23년도에는 2번 보낸 changhee, 22년도에는 1번 보낸 sewha가 최대로 배송을 보낸 사람들입니다. 출력 형식은 customer id, 횟수, name 순으로 하였습니다.

```
nsert into shipment values (77010200, "2023-06-06", "changhee", "coco", 2002, 0001, NULL, 010200);
nsert into shipment values (77030401, "2022-06-05", "sewha", "sogang", 2002, 0003, NULL, 030401);
nsert into shipment values (77030400, "2023-06-05", "sewha", "sogang", 2002, 0003, NULL, 030400);
nsert into shipment values (77050000, "2023-06-06", "star(Coupang)", "star", 3002, 0005, 0005001401, 050000);
nsert into shipment values (77011200, "2023-06-04", "changhee", "google", 2002, 0001, NULL, 011200);
```

```
else if (input == 2)
{
    string year;
    cout << "Find the customer who has shipped the most packages in the past year." << endl;
    cout << "Wich Year? ";
    cin >> year;
    string query = "select customer_id, count(shipment_id) as shipment_count, customer_name from shipment natural join customers where YEAR(pickup_da
    tmpstate = mysql_query(connection, query.c_str());
    if (tmpstate == 0)
    {
        sql_result = mysql_store_result(connection);
        int cnt = 0;
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
        {
            cout << "customer who has shipped the most packages in the " << year << endl;
            cout << "customer_id: " << sql_row[0] << " count: " << sql_row[1] << " name: " << sql_row[2] << endl;
            cnt++;
        }
        mysql_free_result(sql_result);
        if (cnt == 0)
            cout << "There is no shipping data for that year." << endl;
    }
    else
        printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
}
```

```
Find the customer who has shipped the most packages in the past year.
Wich Year? 2023
customer who has shipped the most packages in the 2023
customer_id: 1 count: 2 name: changhee
--------------- SELECT QUERY TYPES ---------------

1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE  V
0. QUIT
2
Find the customer who has shipped the most packages in the past year.
Wich Year? 2022
customer who has shipped the most packages in the 2022
customer_id: 3 count: 1 name: sehwa
--------------- SELECT QUERY TYPES ---------------

1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE  V
0. QUIT
2
Find the customer who has shipped the most packages in the past year.
Wich Year? 2021
There is no shipping data for that year.
```

<구현 코드 및 구현 결과>

3. 2번과 같이 연도를 입력으로 받고, 해당 연도에서 가장 배송으로 돈을 많이 사용한 customer을 찾는 쿼리문입니다. 마찬가지로 가장 많이 돈을 사용한 고객은 유일하다고 생각하고, 다음과 같은 쿼리문을 작성하였습니다.

   SELECT customer_id, sum(price) AS sum_price, customer_name

   FROM shipment natural join customers natural join service

   WHERE YEAR(pickup_date) = year

GROUP BY customer_id

ORDER BY sum_price DESC

LIMIT 1

서비스에 따라 배송 가격이 결정되며, 배송비에 대한 정보는 service table에 있으므로 추가로 natural join을 해줍니다. 마찬가지로 customer의 이름을 구하기 위해 customers table도 natural join을 합니다. 횟수가 아니라 값을 더해주는 sum()을 사용하였다는 점을 제외하면 2번 쿼리와 유사합니다.

```
else if (input == 3)
{
string year;
cout << "Find the customer who has spent the most money on shipping in the past year." << endl;
cout << "Wich Year? ";
cin >> year;
string query = "select customer_id, sum(price) as sum_price, customer_name from shipment natural join customers natural join service where YEAR
tmpstate = mysql_query(connection, query.c_str());
if (tmpstate == 0)
{
    sql_result = mysql_store_result(connection);
    int cnt = 0;
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        cout << "customer who has spent the most packages in the " << year << endl;
        cout << "customer_id: " << sql_row[0] << " sum price: " << sql_row[1] << " name: " << sql_row[2] << endl;
        cnt++;
    }
    mysql_free_result(sql_result);
    if (cnt == 0)
        cout << "There is no shipping data for that year." << endl;
}
else
    printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
}
```

```
3
Find the customer who has spent the most money on shipping in the past year.
Wich Year? 2023
customer who has spent the most packages in the 2023
customer_id: 1 sum price: 4000 name: changhee
--------------- SELECT QUERY TYPES ---------------

1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE  V
0. QUIT
3
Find the customer who has spent the most money on shipping in the past year.
Wich Year? 2022
customer who has spent the most packages in the 2022
customer_id: 3 sum price: 2000 name: sehwa
--------------- SELECT QUERY TYPES ---------------

1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE  V
0. QUIT
3
Find the customer who has spent the most money on shipping in the past year.
Wich Year? 2021
There is no shipping data for that year.
```

<구현 코드 및 결과>

4. 4번 쿼리문은 약속한 도착 날짜에 도착하지 않은 package에 대해 질의하고 있습니다. Package table에 있는 delivery date는 예상 도착 시간을 의미합니다. 해당 값과 located table에 있는 아직 도착하지 않은 수화물이나 예상 도착 날짜를 지난 수화물에 대해 찾으면 됩니다. Date를 기준으로 비교할 예정입니다. 쿼리문은 다음과 같습니다.

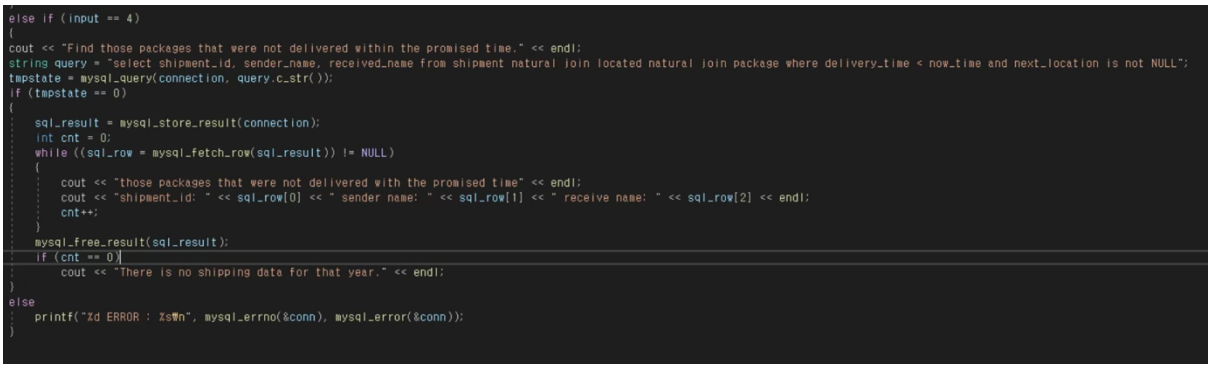
select shipment_id, sender_name, received_name

from shipment natural join located natural join package

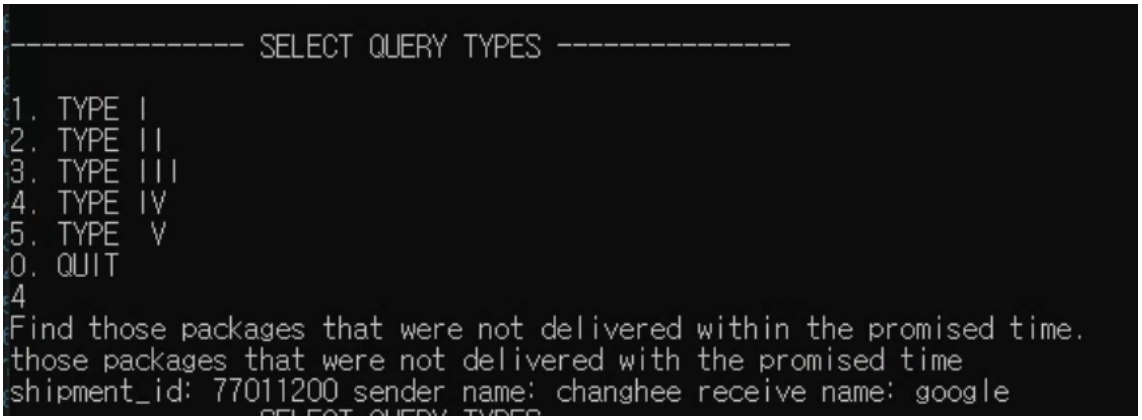
where delivery_date < now_time and next_location is not NULL

```
insert into package values (010200, 400.0, "small box","2023-06-10", "books");
insert into package values (030400, 200.0, "small box","2023-06-07", "puzzle");
insert into package values (030401, 200.0, "small box","2022-06-07", "puzzle");
insert into package values (050000, 550.0, "big box","2023-06-10", "dolls");
insert into package values (011200, 300.0, "small box","2023-06-06", "ipad");

insert into check_package values ('0', '0', 010200);
insert into check_package values ('0', '0', 030400);
insert into check_package values ('0', '0', 030401);
insert into check_package values ('0', '0', 050000);
insert into check_package values ('0', '1', 011200);

insert into shipment values (77010200, "2023-06-06", "changhee", "coco", 2002, 0001, NULL, 010200);
insert into shipment values (77030401, "2022-06-05", "sewha", "sogang", 2002, 0003, NULL, 030401);
insert into shipment values (77030400, "2023-06-05", "sewha", "sogang", 2002, 0003, NULL, 030400);
insert into shipment values (77050000, "2023-06-06", "star(Coupang)", "star", 3002, 0005, 0005001401, 050000);
insert into shipment values (77011200, "2023-06-04", "changhee", "google", 2002, 0001, NULL, 011200);

insert into located values ("truck", 1111, 77010200, "2023-06-08", "Gangdong-gu, Seoul, Republic of Korea", "Songpa-gu, Seoul, Republic of Korea", "lyuminho");
insert into located values ("truck", 1111, 77030401, "2022-06-07", "sogang, 118, Seogang-ro, Mapo-gu, Seoul, Republic of Korea", NULL, "lyuminho");
insert into located values ("truck", 1111, 77030400, "2023-06-07", "sogang, 118, Seogang-ro, Mapo-gu, Seoul, Republic of Korea", NULL, "lyuminho");
insert into located values ("truck", 1111, 77050000,  "2023-06-08", "Gangdong-gu, Seoul, Republic of Korea", "Songpa-gu, Seoul, Republic of Korea", "lyuminho");
insert into located values ("plane", 7777, 77011200, "2023-06-08", "Gangseo-gu, Seoul, Republic of Korea", "Songpa-gu, Seoul, Republic of Korea", "lyuminho");
```

CRUD.txt상으로, 다른 데이터는 해당하지 않지만, 운송번호 77011200인 아이패드를 실은 비행기는 6월 6일 도착 예정이었으나, 6월 8일까지도 도착하지 못한 상황입니다. 따라서 해당 수화물에 대해 출력이 이루어져야 할 것입니다. 출력 형태는 운송번호, 보내는 사람, 받는 사람을 출력하였습니다.



```
else if (input == 4)
{
cout << "Find those packages that were not delivered within the promised time." << endl;
string query = "select shipment_id, sender_name, received_name from shipment natural join located natural join package where delivery_time < now_time and next_location is not NULL";
tmpstate = mysql_query(connection, query.c_str());
if (tmpstate == 0)
{
    sql_result = mysql_store_result(connection);
    int cnt = 0;
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        cout << "those packages that were not delivered with the promised time" << endl;
        cout << "shipment_id: " << sql_row[0] << " sender name: " << sql_row[1] << " receive name: " << sql_row[2] << endl;
        cnt++;
    }
    mysql_free_result(sql_result);
    if (cnt == 0)
        cout << "There is no shipping data for that year." << endl;
}
else
    printf("%d ERROR : %s\n", mysql_errno(&conn), mysql_error(&conn));
}
```

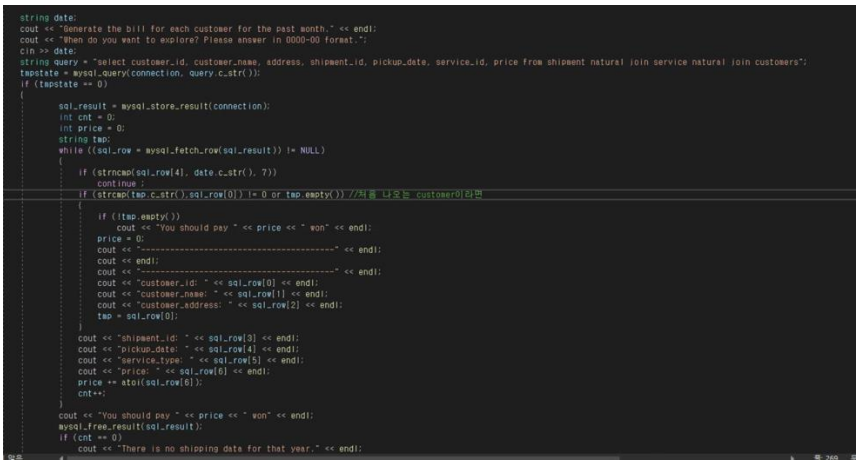


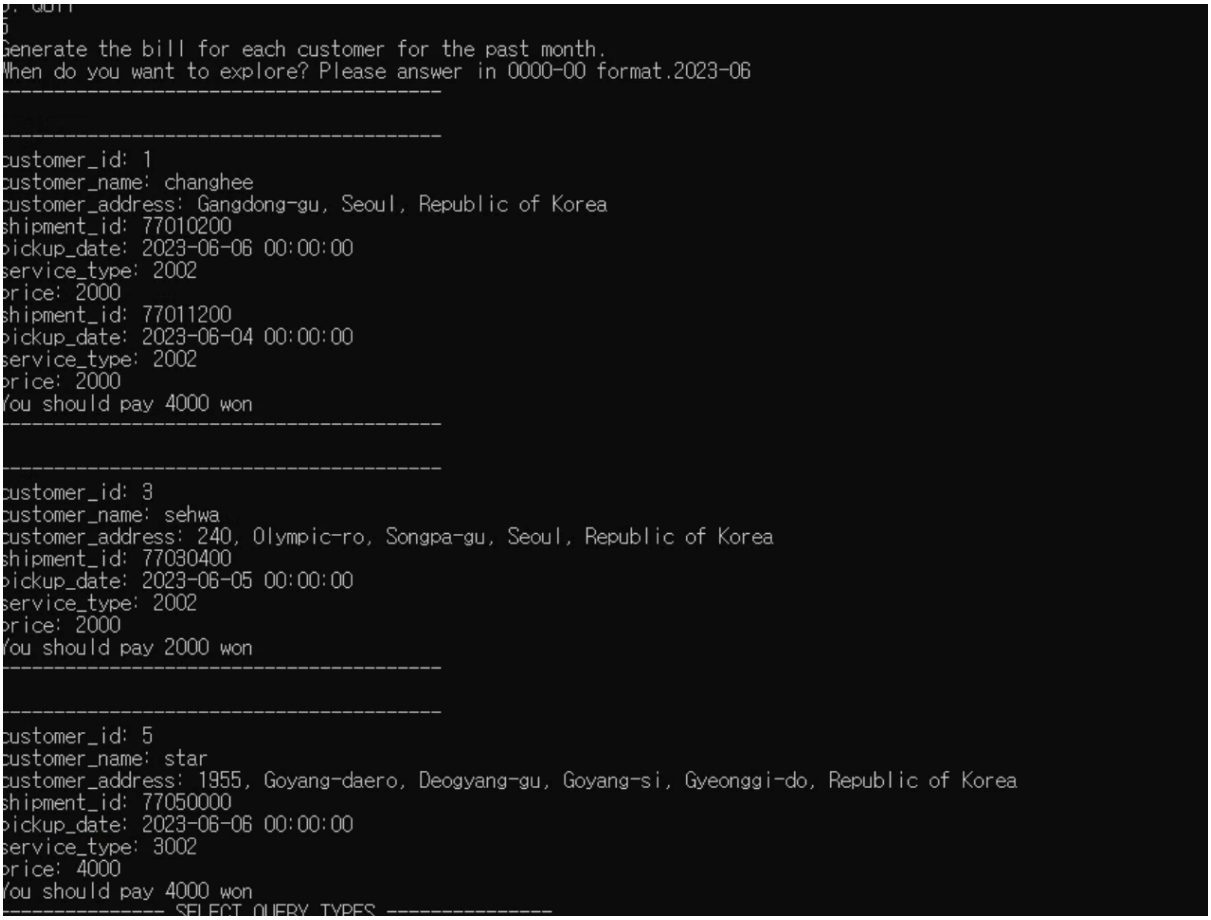
<구현 코드 및 출력 결과>

5. 모든 고객에 대해 청구서를 발급하도록 하였습니다. 입력으로 '0000-00'형식이 오면 입력에 해당되는 달을 출력합니다. 청구서는 고객 id, 고객 이름, 고객 주소, 운송번호, pickup된 날짜, 서비스 타입, 타입에 해당되는 비용이 포함되어있으며, 한 달에 여러 배송이 있었다면, 마지막에 총 배송비를 추가로 첨부합니다. 가독성을 높이기 위해 쿼리문으로 정보를 가져온 뒤, 적절하게 c++언어를 거쳐 출력되도록 하였습니다. 다음은 쿼리입니다.

   select customer_id, customer_name, address, shipment_id, pickup_date, service_id, price

   from shipment natural join service natural join customers




0. Quit
5
Generate the bill for each customer for the past month.
When do you want to explore? Please answer in 0000-00 format.2023-06
customer_id: 1
customer_name: changhee
customer_address: Gangdong-gu, Seoul, Republic of Korea
shipment_id: 77010200
pickup_date: 2023-06-06 00:00:00
service_type: 2002
price: 2000
shipment_id: 77011200
pickup_date: 2023-06-04 00:00:00
service_type: 2002
price: 2000
You should pay 4000 won
customer_id: 3
customer_name: sehwa
customer_address: 240, Olympic-ro, Songpa-gu, Seoul, Republic of Korea
shipment_id: 77030400
pickup_date: 2023-06-05 00:00:00
service_type: 2002
price: 2000
You should pay 2000 won
customer_id: 5
customer_name: star
customer_address: 1955, Goyang-daero, Deogyang-gu, Goyang-si, Gyeonggi-do, Republic of Korea
shipment_id: 77050000
pickup_date: 2023-06-06 00:00:00
service_type: 3002
price: 4000
You should pay 4000 won
SELECT QUERY TYPES

<코드 구현 및 결과>