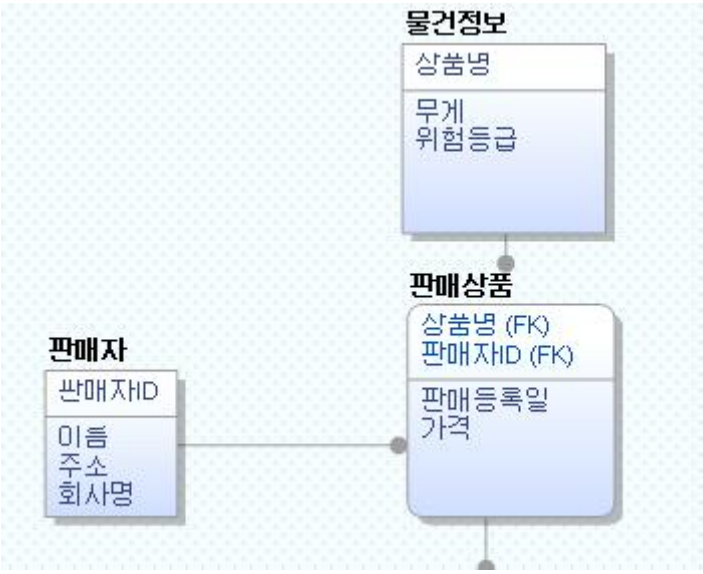
데이터베이스 시스템

20173100 지용환

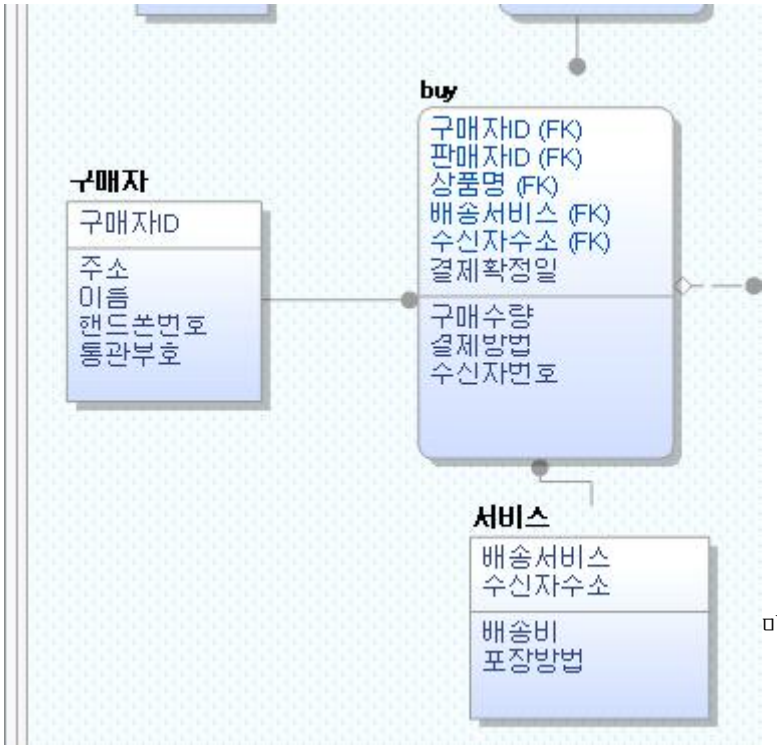
리포트의 순서는 project1에서 relationship들의 변경점들(BCNF를 위해 변경했거나, 그냥 편의상 변경했거나), 환경설정(실행방법), 그 이후 c코드로 구현한 쿼리들순으로 설명을 하겠다.

주목적은 BCNF만들기와 n:n관계를 바꾸고, 각 entity의 relation갯수롤 최소화하는데 목적을 두었다.

먼저 판매자 구매자 entity는 그대로 유지시켰다.

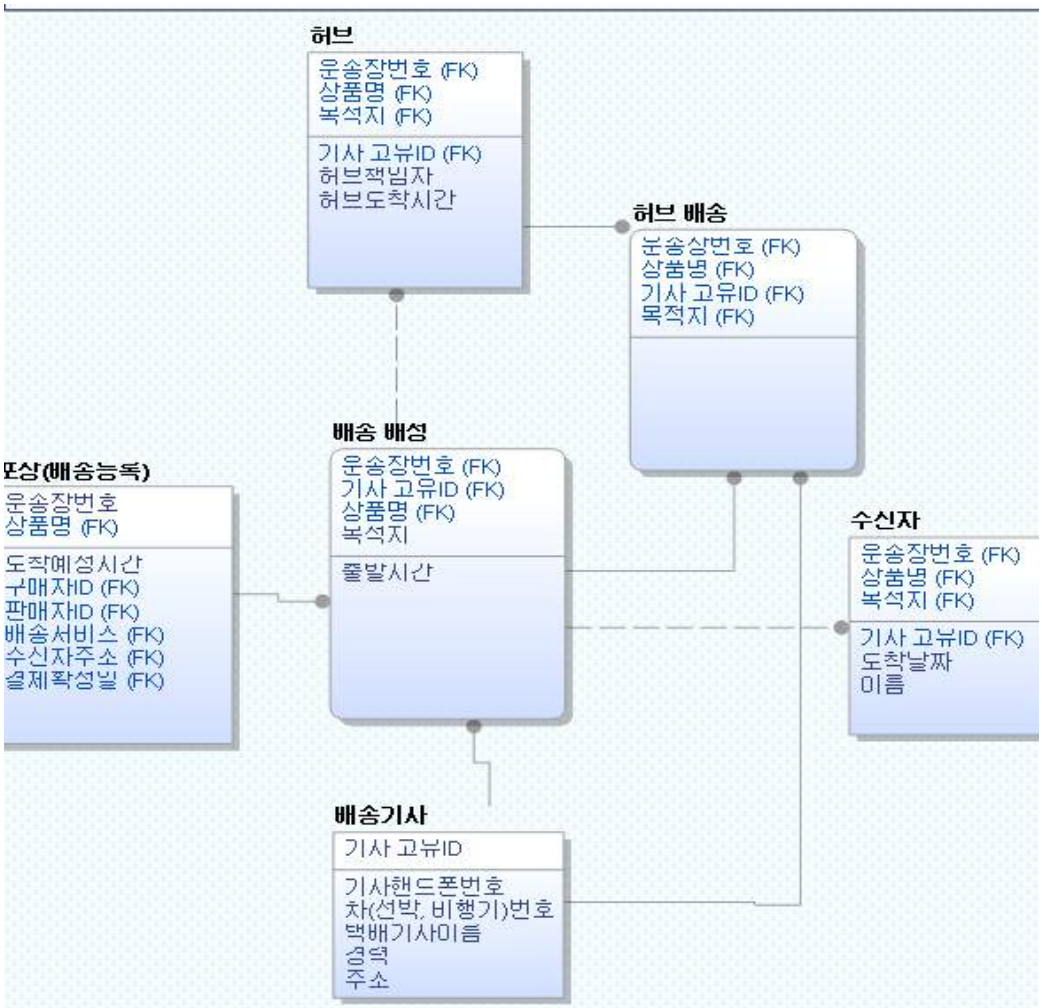


판매상품 entity는 BCNF를 위배했기에 쪼겠는데, 원래 판매상품안에 다 있었던, relation중에 상품명에 따라 무게, 위험등급은 알아서 정해지는 것이기 때문에 따로 물건정보 entity를 만들어 .BCNF 폼을 만들어 주었다.



마찬가지로 buy또한 쪼갰는데, 배송비나 포장방법이, 배송서비스 그리고 수신자의 대략적인 주소에 따라 정해질 것이 분명하기에, (멀고 좋은 서비스면 비싸고(퀵배송?).. 가깝고 나쁜 서비스면 싼것처럼 (동일권 반값택배?)) 이또한 따로 서비스 entity를 추가해 bcnf를 유지할수 있게 해주었다. buy entity 안에 있는 튜플들은, 구매자가 판매상품을 어떤 서비스로 구매했는가에 대한 정보들을 모두 담고 있는 것이다.

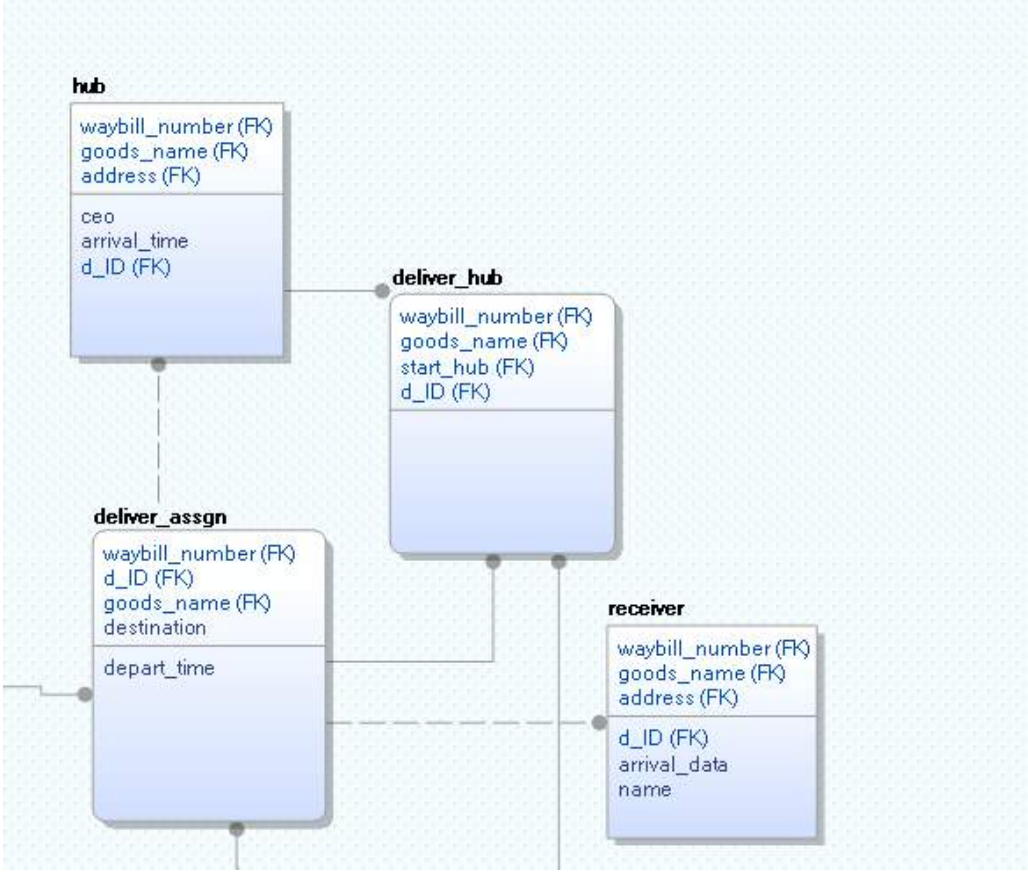
BCNF는 아니지만 변경한 부분인데, 배송등록이란 부부은 판매상품 보다는 buy를 해야지 되는 것이기에 buy와 연결하는 것이 더 좋겠다 판단했고 이에 따라 buy와 연결을 하였다. 한 상품당 운송장번호, 상품명은 unique하게 대응되기 때문에 1:1이고 주목적은 계속 전이되는 너무 많은 외래키를 좀 줄이는 것으로, 이제 운송장 번호와 상품명이 primariy key가 되는 것이다. 그 외엔 거의 비슷하다.
포장 entity에 들어왔다면, 판매자가 상품을 배달회사에 맡겼다고 봐도 무방하다.



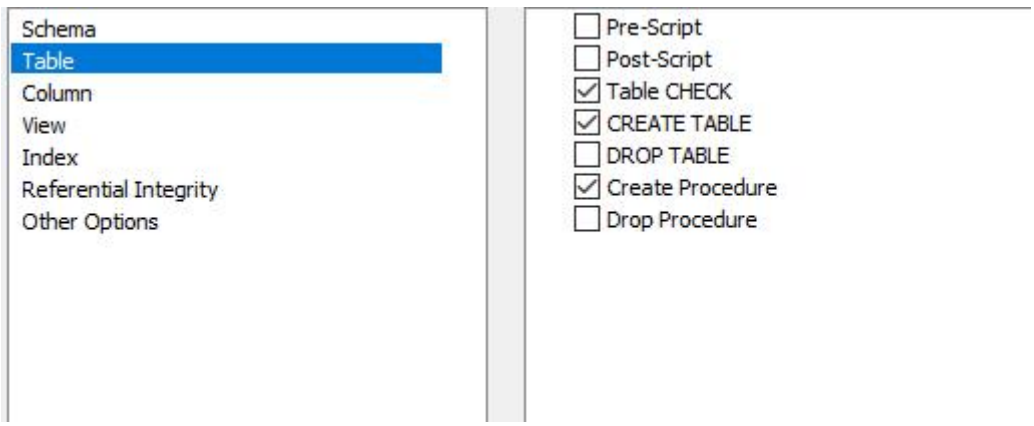
가장 많이 수정을 한 부분인데, 이 부분이 배송관련 거의 모든 정보를 담고 있다. 여기선 BCNF만들기 보단 n:n의 관계를 깨는데 주 목적을 두었다. 먼저 배송기사 entity는 기사 고유 ID를 추가해 그냥 가장 단순한 pk를 만들어 외래키로 너무 많은 키가 전이되는걸그냥 막았고, 배송배정은 1차적으로 포장, 그리고 배송기사의 배정이 필요하기에 이 들의 키를 슈퍼키로 간주하게 해 주었다. (애초에 1:1 과 1:n이기도 하고) 배송배정은 추가적으로 목적지와, 출발시간을 보유하고 있고. 이는 hub entity나 수신자 entity로 간다. 실제로 hub나 수신자는 굉장히 유사한 entity인데, pk가 동일하고, 이들에게서의 목적지는 hub의 주소, 혹은 수신자의 주소이다. 추가적인 것은 도착시간 그 외 부가 정보들을 각각 추가해 주었다. 수신자에게 갔다면 종료되고, 만약 hub로 갔다면 추가적으로 다시 재 배정이 필요하기 때문에 허브배송이란 entity를 추가해주었다. 이때 허브배송은

physical에선 이렇게 이름을 바꿔주었는데, 가각 운송장번호, 상품명,

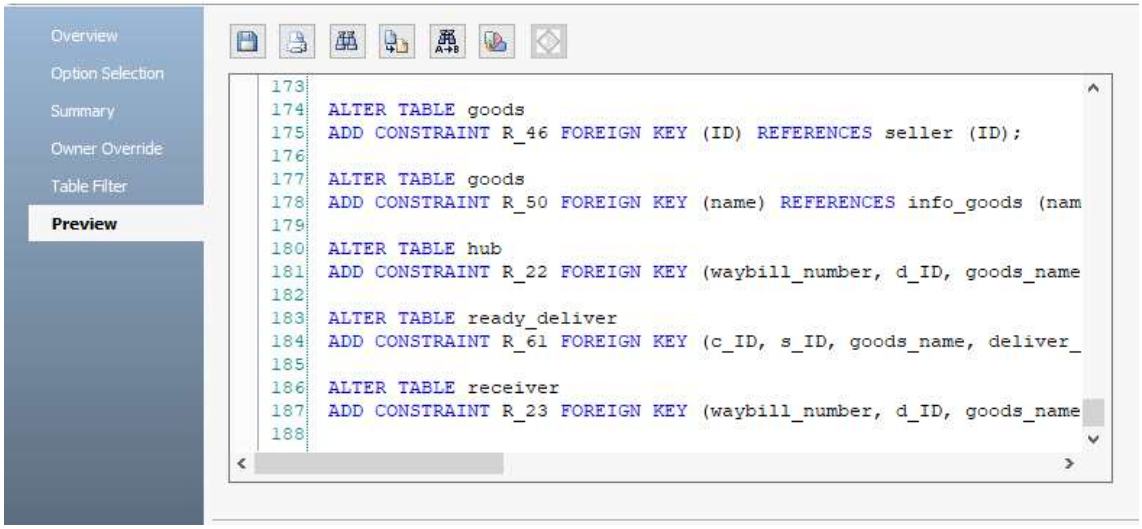
출발허브위치, 운전자 id이다. 간단히 hub에서 새로운 운전자를 배정받는 것이다. 이렇게 되면 다시 배송배정 eneity로가 새로운 출발시간과 새로운 목적지를 가지게 해주었다. 실제로 위의 관계들을 physical 로 바꿀 때 더 명확한 이름을 주어 구분이 가능하게 해주었는데 다음과 같다.



실행은 단순히, physical schema에서 모두 영어이름으로 바꾼후,



action->schema, 그리고 모두 기본값으로 진행해



다음과 같이 만들어진 코드를 mysql에 복붙해서 사용했다.(파일을 읽는데, 이상하게 공백들이 인식이 잔안되서 저부분만 수정을 했다.

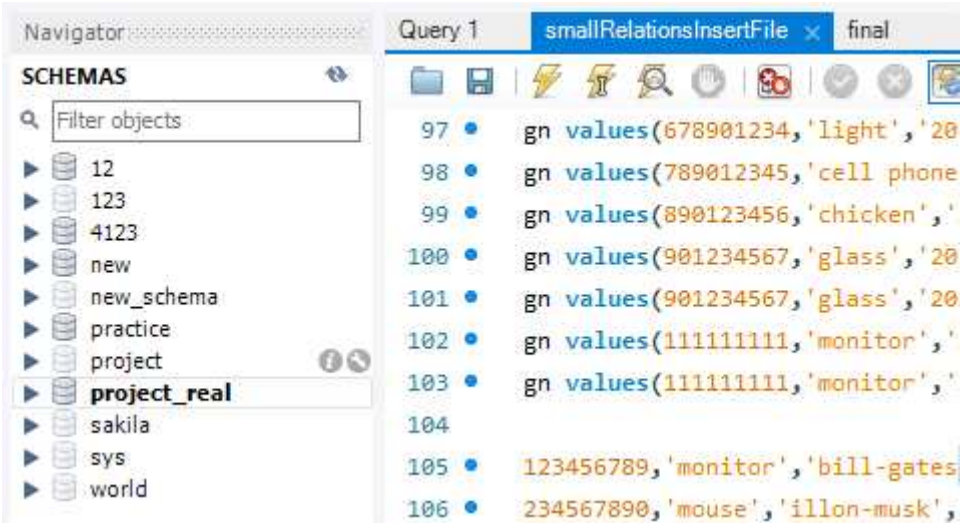
ex :

ALTER TABLE ...

ADD .... 를

ALTER TABLE .. ADD ..로 \N만 삭제)

그이후, mysql에서 스키마 생성후(project_real에서 진행함)



위의 복붙 코드로 table생성, 그리고 데이터들을 한 테이블당 약 10개 정도씩 추가해 실험을 진행했다.

마지막으로 DROP파일을 읽어 모든 TABLE을 DROP하며 종료시켜주었다.

```sql
drop table receiver;
drop table deliver_hub;
drop table hub;
drop table deliver_assgn;
drop table ready_deliver;
drop table driver;
drop table buy;
drop table service_buy;
drop table goods;
drop table info_goods;
```

이 모든 CRUD파일들은 creat.sql, insert.sql, drop.sql의 파일로 각각 저장이 되어있고, create, insert는 코드 실행하자마자 읽으며 실행. 그리고 drop은 코드의 마지막에 실행을 한다.
즉, 강제 종료를 하거나 잘못된 input을 주어서 강제 종료가 된다면 drop이 되지 않기 때문에 꼭 0을 입력해 종료하는 것을 추천한다.
간단히 create파일 읽는 부분만 설명하면,

```cpp
mysql_init(&conn);
if (!mysql_real_connect(&conn, host, user, pw, db, 0, NULL, 0)) {
    cerr << "MySQL 연결 실패: " << mysql_error(&conn) << endl;
    return 1;
}

string filename = "create.sql";
ifstream file(filename);

if (!file.is_open()) {
    cerr << "파일 열기 실패" << endl;
    return 1;
}

string line;
string query;
while (getline(file, line)) {
    query += line;
    // 쿼리 종결자인 ';'를 기준으로 한 쿼리씩 분리하여 실행
    if (!line.empty() && line.back() == ';') {
        if (mysql_query(&conn, query.c_str()) != 0) {
            cerr << "쿼리 실행 실패: " << mysql_error(&conn) << endl;
            return 1;
        }
        query.clear();
    }
}

file.close();
mysql_close(&conn);
```

코드는 다음과 같고, ;기준으로 한 쿼리씩 실행을 할수 있도록 해준 것이다. 다읽으면 성공적으로 결과가 반영되게 close를 시켜버렸고, 그이후 다시 mysql_init을 하여 다시 그 sql을 읽는 과정을 insert에서 한다.

마지막으로 설명을 할 것은 코드 부분이다. 전역함수 먼저 설명하면,

```cpp
//datetime1이 더 크면, 그니깐, 더 늦거나 같은 시간일시, true
bool compareDatetime(string datetime1,string datetime2) {
```

다음과 같은 함수 하나를 추가했는데, 이유는 단순히 쿼리 생각하는게 너무 어지러 그냥 c++의 라이브러

리를 활용해 좀더 간단한 쿼리로, 입력을 받은후 그 이후 datetime을 비교하기 위한 함수이다. datetime1이 더 늦거나 같은 시간이면 true를 반환하게 해주었다.

option1: 옵션 1은 3개의 서브 opntio이 있는데, 이 3개의 서브 옵션을 받기전, 존재하는 차임을 검사하는 과정을 추가했다.

```
132                 printf(" Which truck? : ");
133                 cin >> car_number;
134                 query = "SELECT * FROM driver WHERE car_number = "+car_number;
135                 int state = 0;
136                 state = mysql_query(connection, query.c_str());
137
138                 if (mysql_query(connection, query.c_str()) == 1) {
139                     sql_result = mysql_store_result(connection);
140                     if ((sql_row = mysql_fetch_row(sql_result)) == NULL) {
141                         cout << "no car\n";
142                     }
```

단순히 car_number조회를 실패할시 no car를 출력하고 메인메뉴로 돌아간다.

```
**  Assume truck X is destroyed in a crash.**
 Which truck? : 0000
no car
------- SELECT QUERY TYPES -------

        1. TYPE 1
        2. TYPE 2
        3. TYPE 3
        4. TYPE 4
        5. TYPE 5
        0. QUIT
```

그 외에 차가 존재한다면, 0을 입력전까지 계속 반복해 서브쿼리를 입력받도록 whil을 활용했고, 이때 차가 파괴되는 시간까지 입력을 하게 해주었다. 파괴되는 시간은 반드시 datetime형식으로 해주어야 하며, 그렇지 않다면 에러가 난다.

```
else {
    mysql_free_result(sql_result);
    while(1)
    {
        printf("\n\n");
        printf("---- Subtypes in TYPE 1 ----\n\n");
        printf("    1. TYPE 1.1.\n");
        printf("    1. TYPE 1.2.\n");
        printf("    1. TYPE 1.3.\n");
        cin >> option2;
        if (option2 == 0) { ... }
        printf("input car destroyed date (yyyy-mm-dd hh:mm:ss) : ");
        cin >> year;
        cin >> time;
        year = year + " " + time;
        query = "SELECT deliver_assgn.waybill_number, deliver_assgn.depart_time, deliver_assgn.destination, hu
            "From deliver_assgn LEFT OUTER JOIN hub ON deliver_assgn.waybill_number =hub.waybill_number AND de
            "LEFT OUTER JOIN receiver ON deliver_assgn.waybill_number = receiver.waybill_number AND deliver_as
            "LEFT OUTER JOIN driver ON deliver_assgn.d_id =driver.id "
            "LEFT OUTER JOIN ready_deliver ON deliver_assgn.waybill_number =ready_deliver.waybill_number "
            "ORDER BY deliver_assgn.depart_time";
        int state = 0;
```

위의 쿼리는 deliver_assgn기준으로 left outer join을 한것인데, 배송배정 기준 수신자, 허브, divver, 배송준비를 join하여 정보를 얻었다. 이때 outer join을 한 이유는 배송배정 기준으로 수신자로 갈수도

있고 허브로 갈수도 있기 때문에 null값이 존재해야만 했었고, 그렇기에 outer join을 해서 그 이후 시간비교를 하려고 했다. 가장먼저 한 일은, 어떠한 차가 배송한 모든 운송장 번호 데이터를 검색한 것이다.

```
while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
{
    if (sql_row[5] == car_number) {
        //그 차를 거쳐간 모든 운송장 번호들 벡터
        car_waybill.push_back(sql_row[0]);
        if (sql_row[3]) {
            arrival_time.push_back({ sql_row[3],sql_row[0] });
        }
        else if (sql_row[4]) {
            arrival_time.push_back({ sql_row[4],sql_row[0] });
        }
    }
```

추가적으로 arrival_time이란 벡터도 추가해 도착시간,운송장 번호를 pair로 가지게 해주었다. 이 벡터는 null값을 먼저 조사해서, sql_row[3]이 null이면 sql_row[4]를 추가하는 즉 수신자에게 도착한 것이면 sql_row[3]이 null이니 수신자 도착시간인 sql_row[4]를 도착시간에 추가한 벡터이다.

```
**  Assume truck X is destroyed in a crash.**
 Which truck? : 0480


---- Subtypes in TYPE 1 ----

        1. TYPE 1.1.
        1. TYPE 1.2.
        1. TYPE 1.3.
1
input car destroyed date (yyyy-mm-dd hh:mm:ss) : 2023-06-10 19:19:19
```

이후에 다시 쿼리를 실행, 그리고 검사를 했는데
option 1-1 :

```
while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
{
    if(find(car_waybill.begin(),car_waybill.end(),sql_row[0])!=car_waybill.end())
    {
        //hub에게
        if (sql_row[3] &&car_number==sql_row[5]) {
            if (compareDatetime(year, sql_row[1]) && compareDatetime(sql_row[3], year)) {
                printf("%s's(customer) goods in %s at %s (this truck start deliver at %s)\n",sql_row[6],
                count++;
            }

        }
        //수신자에게
        else if(sql_row[4] && car_number == sql_row[5]) {
            if (compareDatetime(year, sql_row[1]) && compareDatetime(sql_row[4], year)) {
                printf("%s's(customer) goods in %s at %s (this truck start deliver at %s)\n", sql_row[6],
                count++;
            }
        }
```

다음과 같은 코드로 입력한 시간이 sql_row[1]즉 배송 출발시간보단 크고, sql_row[3,4]즉 도착시간보다 작은 경우, 충돌시에 그 트럭안에 있었다는 것으로 판단해 결과를 출력시켜주었다.

```
---- Subtypes in TYPE 1 ----

        1. TYPE 1.1.
        1. TYPE 1.2.
        1. TYPE 1.3.
1
input car destroyed date (yyyy-mm-dd hh:mm:ss) : 2023-06-10 19:19:19
 result is:
a's(customer) goods in 0480 at 2023-06-10 19:19:19 (this truck start deliver at 2023-06-10 04:00:00)
a's(customer) goods in 0480 at 2023-06-10 19:19:19 (this truck start deliver at 2023-06-10 04:00:00)
e's(customer) goods in 0480 at 2023-06-10 19:19:19 (this truck start deliver at 2023-06-10 10:00:00)
```

위의 결과는 출력의 결과로,

insert into deliver_assgn values(234567890,'mouse','2023-06-10 20:00:00','hubman2','gimpohub'); ~

insert into hub values(234567890,'mouse','bill-gates','2023-06-11 01:30:00','hubman2','gimpohub');

insert into deliver_assgn values(345678901,'water','2023-06-10 04:00:00','hubman2','gimpohub'); ~

insert into hub values(345678901,'water','bill-gates','2023-06-11 03:20:00','hubman2','gimpohub');

insert into deliver_assgn values(111111111,'monitor','2023-06-10 10:00:00','hubman2','gimpohub'); ~

insert into hub values(111111111,'monitor','bill-gates','2023-06-10 20:00:00','hubman2','gimpohub');

위의 총 3개의 결과가 출력된 것이다. 각각 출발시간, 도착시간 의 정보를 가지고있는 tuple이고 진한색으로, 밑줄로 표시했다.

option 1-2: 옵션 2의 주목적은 1과 비슷한데, 소비자가아닌 수신자의 정보를 표시하는 것이다.

```
222         vector<string> semi_result;
223         while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
224         {
225             if (find(car_waybill.begin(), car_waybill.end(), sql_row[0]) != car_waybill.end())
226             {
227                 //허브 물품 시간 체크(여기서 체크가 돼도, 수신자를 찾을수 있도록 semi_result에 추가)
228                 if (sql_row[3] && car_number == sql_row[5]) {
229                     if (compareDatetime(year, sql_row[1]) && compareDatetime(sql_row[3], year))
230                     {
231                         semi_result.push_back(sql_row[0]);
232                     }
233                 }
234                 //수신자 전송 시간 체크
235                 else if (sql_row[4] && car_number == sql_row[5]) {
236                     if (compareDatetime(year, sql_row[1]) && compareDatetime(sql_row[4], year))
237                     {
238                         semi_result.push_back(sql_row[0]);
239                     }
240                 }
241                 //트럭 폭발시 운송장 번호를 가지고 있는 semi_result의에서..
242                 if (find(semi_result.begin(), semi_result.end(), sql_row[0]) != semi_result.end()) {
243                     //수신자 정보 찾을수 있는 tuple보면 출력
244                     if (sql_row[4]) {
245                         printf("%s's(receiver) goods in %s at %s (receive at %s)\n", sql_row[7], car_num
246                         count++;
247                     }
248                 }
249             }
250         }
251     }
```

주석을 달아놓긴했는데 설명하면, 1번과 비슷하지만, 먼저 폭발했을시 트럭에 있었던 물품들을 semi_result란 벡터에 추가를 한후, 모든 튜플에대해 해당 운송장번호를 받은 수신자 정보가 null이 아니라면, 그것을 출력하는 형식으로 코드를 짰다.

결과는 다음과 같고, 실제로 1의 결과 3개중 2개의 값이 있는데, 이는 receive 정보에 마지막 물건의 정보를 넣지 않았기 때문이다. 괄호안의 내용은, 충돌이 일어나지 않았더라면 원래 도착하는 시간을 표시해주었다.

option1-3: 마지막 옵션의 목적은, 충돌 전 가장 마지막으로 배송에 성공한 물건정보를 출력하는 것이다. 이를 구현하기 위해 먼저

```
//arrival_time낮은것 부터 정렬.
sort(arrival_time.begin( ), arrival_time.end( ));
//충돌시간보다 작은것중 가장 큰거. 그니깐 year보다 작으면 계속
string last_day="0000-00-00 00:00:00";
string zz="";
for (int i = 0; i < arrival_time.size( ); i++) {
    if (compareDatetime(year, arrival_time[i].first)) {
        last_day = arrival_time[i].first;
        zz = arrival_time[i].second;
    }
    else {
        break;
    }
}
if (zz == "") {
    count = 0;
}
else
{
    count = 1;
```

트럭이 몰았던 모든 운송장 정보와 도착시간 정보를 담고있는 ariival_time을 정렬한 이후, last_dat란 변수를 계속 입력값과 비교하며, 충돌 시점보다 작지만, 그중 가장 큰 것을 담을수 있도록 했다. (이때 같은것도 허락된다.) 이후, update된 운송장 번호와, 시간을 가지고

```
else
{
    count = 1;
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
    {
        if (sql_row[0] == zz) {
            if (sql_row[3] && sql_row[3]==last_day) {
                printf("If destroy at %s then last successful deliver of %s(truck) is %s.(Success a
                break;
            }
            else if (sql_row[4] && sql_row[4] == last_day){
                printf("If destroy at %s then last successful deliver of %s(truck) is %s.(Success a
                break;
            }
        }
    }
}
```

튜플을 탐색하여 같은 것을 찾을시, 그것을 출력하고 탈출할수 있도록 했다.

```
3
input car destroyed date (yyyy-mm-dd hh:mm:ss) : 2023-06-10 20:00:01
 result is:
If destroy at 2023-06-10 20:00:01 then last successful deliver of 0480(truck) is 11111111.(Success at 2023-06-10 20:00:00)
```

결과는 다음과 같은데, 충돌전 도착시갆중 가장 마지막것의 물건 정보, 그리고 그것의 도착시간을 가르쳐 준다. 11111111은 해당 물품의 운송장 번호이다.

```
        }
        if (count == 0)
            cout << "none₩n";
        mysql_free_result(sql_result);
        arrival_time.clear( );
    }
}
```

이 문장은 마지막에 만약 옵션 1,2,3중 하나 아무거나 선택했을 때, 결과가 하나도 없다면 count=0이 되게 했는데, 그러면 none을 출력하는 것이다.

```
input car destroyed date (yyyy-mm-dd hh:mm:ss) : 1999-10-10 10:10:10
 result is:
none
```

option2: 옵션2의 목적은 특정 해에 가장 많이 주문을 한 사람을 뽑은 것인데, 단순하게 buy tuple만을 사용해 작성가능했다.

```
307             query = "SELECT c_id, COUNT(*) AS count FROM buy WHERE YEAR(payment_data) = ";
308             query += year;
309             query += " GROUP BY c_id ORDER BY count DESC LIMIT 1";
310             int state = 0;
311             state = mysql_query(connection, query.c_str( ));
312
313             if (state == 0)
314             {
315                 cout << "At " << year << " most shipped customer is: ";
316                 sql_result = mysql_store_result(connection);
317                 while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
318                 {
319                     printf("%s, %s₩n", sql_row[0], sql_row[1]);
320                     count++;
321                 }
322                 if (count == 0) {
323                     printf("none₩n");
324                 }
325                 mysql_free_result(sql_result);
326             }
327             else {
328                 cout << "error₩n";
329             }
330             printf("₩n");
```

쿼리는, buy에서 특정해 기준으로 선택을 해 id기준 그룹으로 나누어 그것을 count한 것을 select한 것으로 가장 많은 tuple하나만 선정하기위해 내림차순 정렬 중 가장위 limit1을 해서 선택했다. 이렇게 되면 단순히 그 해당해에 가장많이 산 id가 몇 개를 샀는지까지 출력해 준다.

```
---- TYPE 2 ----

** Find the customer who has shipped the most packages in the certain year.**
 Which year? : 2023
At 2023 most shipped customer is: a, 4s
```

```
** Find the customer who has shipped the most packages in the certain year.**
 Which year? : 2022
At 2022 most shipped customer is: b, 2s
```

option3: 2번과 비슷하지만 가장 비싼 금액을 지불한 사용자를 찾는 옵션이다. 금액은 상품의 가격, 상품의 개수 그리고 배송비를 합해야 결정이 되므로 join이 필요했고 해당 쿼리는 다음과 같다.

```
338             query =
339                 "SELECT b.c_ID, SUM(sb.deliver_price + g.price*b.quantity) AS total_payment  "
340                 "FROM buy b "
341                 "JOIN goods g ON b.goods_name = g.name AND b.s_ID=g.ID "
342                 "JOIN service_buy sb ON b.deliver_type = sb.deliver_type AND b.receiver_address = sb.receiver_ad
343                 "WHERE YEAR(b.payment_data) = '" + year + "'"
344                 "GROUP BY b.c_ID "
345                 "ORDER BY total_payment DESC "
346                 "LIMIT 1 ";
347             int state = 0;
```

buy를 기준으로 물건가격이 있는 goods와 배송비 가격이 있는 service_buy를 join해 주었고, 제약 사항으로 year를 주었다. 그후 사용자 별로 그룹을 지어주어서, 물건가격*물건 개수 + 배송비 의 총 합 즉 sum을 select했고, 이또한 내림차순 상위 1개만 뽑아서, 가장 많은 금액을 지불한 사용자를 찾을수 있게했다.

```
** Find the customer who has spent the most money in the past year.**
 Which year? : 2023
At 2023 most spent money customer is: e (spent 404000won)
```

```
** Find the customer who has spent the most money in the past year.**
 Which year? : 2022
At 2022 most spent money customer is: c (spent 1010000won)
```

출력방법은 option2와 거의 동일하기에 생략하겠다.

option4: 약속시간안에 배송이 성공한 물품 모두 찾는 것이다. 약속시간은 ready_deliver안에 존재하고, 수신자에게 도착한 시간은 receiver안에 존재하므로, 그리고 이 tuple은 동일하게 운송장번호, 상품명을 가지고 있기 때문에 natural join을 시켜주었다. 만약 수신자 tuple에 존재하지 않는것이면 어짜피 도착안한것이기 때문에 join을 하지 않는 natural join을 선택했다.

```
371             printf("** Find those packages that were not delivered within the promised time..**\n");
372             query = "SELECT ready_deliver.c_id, ready_deliver.goods_name, ready_deliver.expected_arrival, r
373                 "From ready_deliver NATURAL JOIN receiver "
374                 "WHERE ready_deliver.expected_arrival >= receiver.arrival_data";
375             int state = 0;
376             state = mysql_query(connection, query.c_str());
377
378             if (state == 0)
379             {
380                 cout << " result is:  \n";
381                 sql_result = mysql_store_result(connection);
382
383                 while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
384                 {
385                     //if(compareD(one,two))
386                     printf("%s's %s arrived at %s (expected: %s)\n", sql_row[0], sql_row[1], sql_row[3], s
387                     count++;
388
389                 }
390                 if (count == 0)
391                     cout << "none\n";
392                 mysql_free_result(sql_result);
393             }
394             else {
395                 cout << "error\n";
396             }
397             printf("\n");
```

쿼리는 다음과 같고, join이후에, datetime을 where절에서 비교해 더 빨리 도착한 모든 tuple을 뽑아주었고, 그것을 그냥 출력해 주었다.

```
---- TYPE 4 ----

** Find those packages that were not delivered within the promised time..**
 result is:
a's monitor arrived at 2023-06-06 12:27:00 (expected: 2023-06-06 20:00:00)
a's mouse arrived at 2023-06-12 16:34:00 (expected: 2023-06-12 19:00:00)
b's pencil arrived at 2022-05-09 22:49:00 (expected: 2022-05-10 16:00:00)
b's light arrived at 2022-04-10 16:30:00 (expected: 2022-04-10 17:00:00)
d's chicken arrived at 2023-04-06 10:30:00 (expected: 2023-04-06 11:00:00)

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

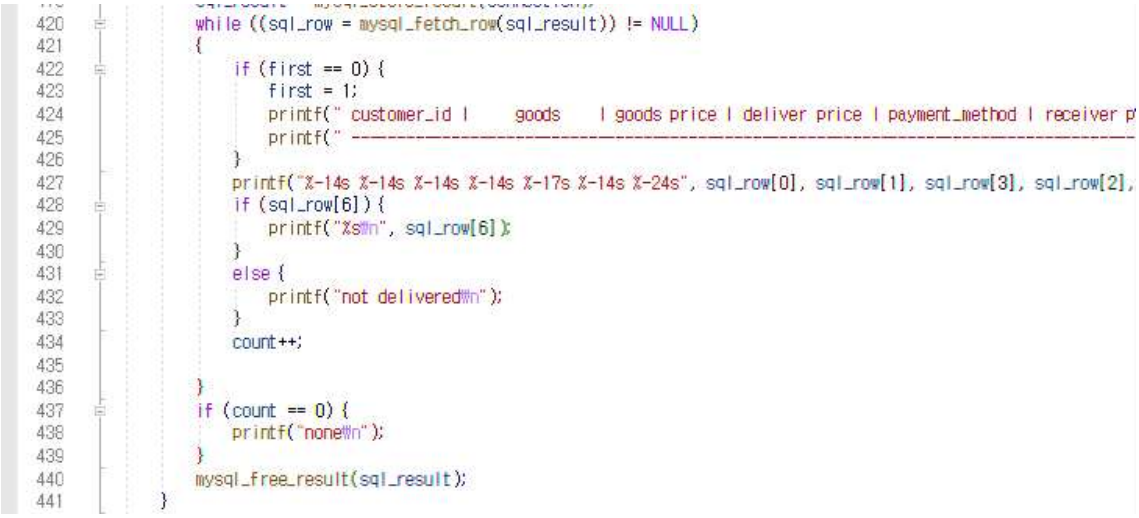
option5: 사용자의 bill을 출력해 주는 것이다. 총 5개의 쿼리를 종합해, relation을 뽑아주었고, 년, 월을 입력받는다. 먼저 결과는 다음과 같다.

```
** Bill at year-month..**
 Input year and month(split by ' ') : 2023 06
 customer_id |    goods    | goods price | deliver price | payment_method | receiver phone |     pyament date     | arrival_date
-------------------------------------------------------------------------------------------------------------------------------
a            chicken       5000          10000           advance pay      01077777777      2023-06-08 06:04:01     2023-06-09 16:13:00
a            monitor       100000        5000            advance pay      01012341234      2023-06-05 06:00:01     2023-06-06 12:27:00
a            mouse         10000         1500            deferred pay     01015971597      2023-06-06 06:02:01     2023-06-12 16:34:00
a            water         50000         3500            regular pay      01012341234      2023-06-07 06:08:01     2023-06-14 07:12:00
d            pencil        20            20000           advance pay      01072048902      2023-06-10 06:20:01     not delivered
e            monitor       200000        4000            deferred pay     01098783210      2023-06-07 06:17:01     not delivered
h            monitor       200000        5000            advance pay      01098183210      2023-06-09 06:18:01     not delivered
```

큰 계산은 없고, 그냥 join해서 소비자, 상품, 가격, 배송비, 결제방법, 수신자 번호, 결제일, 도착일을 그냥 출력했다.

```
406             //buy.c_id, buy.goods_name, goods.price, service_buy.deliver_price, buy.deliver_type, buy.payment
407             query = "SELECT buy.c_id, buy.goods_name, service_buy.deliver_price , goods.price , buy.payment_m
408                 "From buy LEFT OUTER JOIN service_buy ON buy.deliver_type =service_buy.deliver_type AND buy.r
409                 "LEFT OUTER JOIN customer ON buy.c_id = customer.id "
410                 "LEFT OUTER JOIN goods ON buy.s_id =goods.id AND buy.goods_name=goods.name "
411                 "LEFT OUTER JOIN ready_deliver ON buy.c_id =ready_deliver.c_id AND buy.s_id =ready_deliver.s_
412                 "LEFT OUTER JOIN receiver ON ready_deliver.waybill_number =receiver.waybill_number AND receiv
413                 "WHERE YEAR(buy.payment_data) = '" + year + "' AND MONTH(buy.payment_data) = '"+month+ "'";
414             int state = 0;
415             int first = 0;
```

쿼리도 그냥 outer join을 연속적으로 했고,



```
420         while ((sql_row = mysql_fetch_row(sql_result)) != NULL)
421         {
422             if (first == 0) {
423                 first = 1;
424                 printf(" customer_id |     goods     | goods price | deliver price | payment_method | receiver p
425                 printf(" ------------------------------------------------------------------------------------------
426             }
427             printf("%-14s %-14s %-14s %-14s %-17s %-14s %-24s", sql_row[0], sql_row[1], sql_row[3], sql_row[2],
428             if (sql_row[6]) {
429                 printf("%s\n", sql_row[6]);
430             }
431             else {
432                 printf("not delivered\n");
433             }
434             count++;
435 
436         }
437         if (count == 0) {
438             printf("none\n");
439         }
440         mysql_free_result(sql_result);
441     }
```

만약 도착을 안했을시 not delivered 그 외 도착시간을 그냥 포맷팅을 통해 출력하게 해주었다.