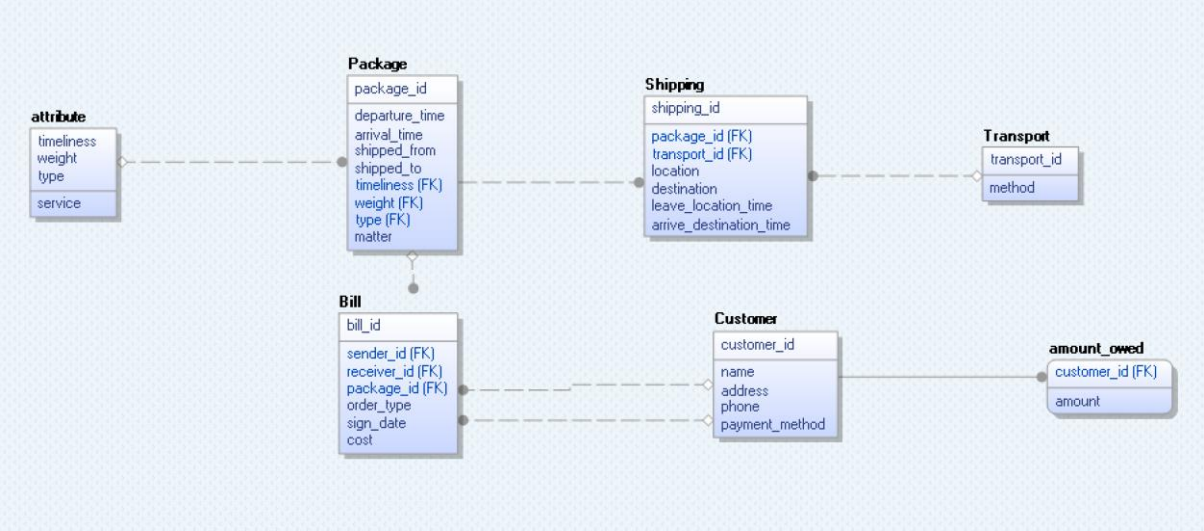
DB project2 보고서

20191638 임형준

1. BCNF decomposition and change point



수정 및 분해가 완료된 logical schema는 위와 같다.

attribute entity set에서 functional dependency는 { timeliness, weight,type_package ->service} 이다 . timeliness, weight,type_package 는 attribute의 superkey이므로 attribute entity set은 bcnf를 만족하는 form이다.

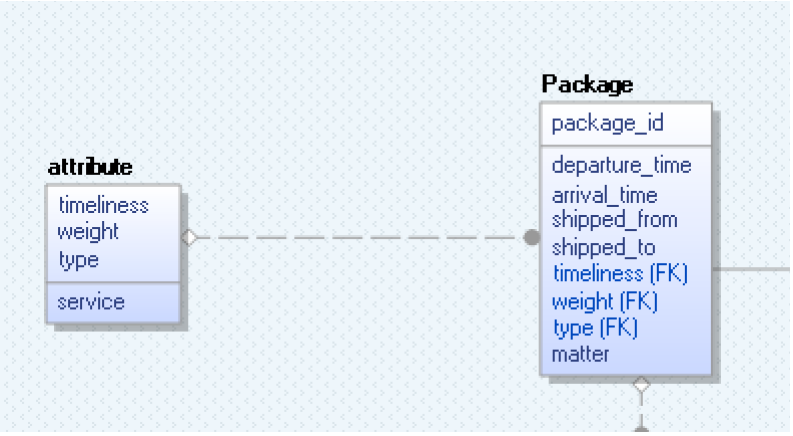
현재 package의 functional dependency를 다음과 같이 표현할 수 있다.

{ package_id -> (timeliness, type, weight, departure_time, arrival_time, shipped_from, shipped_to, service) , (weight, type, timeliness) -> service}

bcnf의 조건인 a -> b 일 때 a는 R의 superkey여야 한다는 조건에 비춰볼 때,

(weight, type, timeliness) -> service 에서 dependency의 왼쪽이 superkey가 아니므로 bcnf를 만족하지 않는다.

이를 위해 decomposition을 진행하였고 다음과 같다.



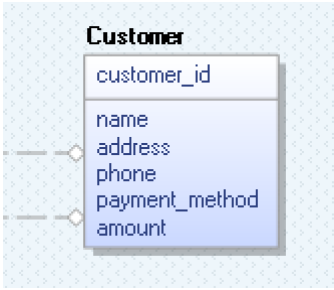
두 개의 relation으로 decompose하여 bcnf를 만족하도록 하였다.

또한 쿼리를 위해 수정된 점이 있는데,

먼저 위의 package에는 extra_info entity set을 삭제하고 matter라는 속성을 추가하여

국제 화물인지 위험한 화물인지를 작성하도록 저장하였다.

현재 작성하고 있는 query에서 딱히 조회되지 않는 내용이기에 간단하게 구현하였다.



Customer에 payment_method와 amount(owed) 가 추가되었다.

functional dependency는 {customer_id -> name, address, phone, payment_method, amount} , {payment_method -> amount} 이다.

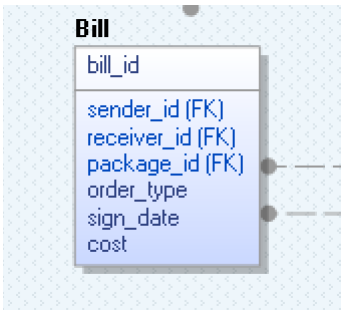
payment_method는 card, account, phone 등 결제 수단을 저장하는 속성이며, 이를 통해 월

별 결제인 지, 혹은 일시불인 지를 결정할 수 있으며 미지불 금액인 amount를 저장하게 된다.

하지만 이는 곧 payment_method가 미지불 금액이 null 일지 not null일 지를 결정한다는 뜻이다 payment_method는 superkey가 아니므로, bcnf를 위반한다.

이렇게 분해하게 되면 functional dependency는 {customer_id -> name, address, phone, payment_method} 이고 customer_id는 Customer의 superkey이므로 bcnf form을 만족한다.

amount_owed의 functional dependency는 {user_id -> amount} 이고 user_id는 amount_owed의 superkey이므로 bcnf form을 만족한다.

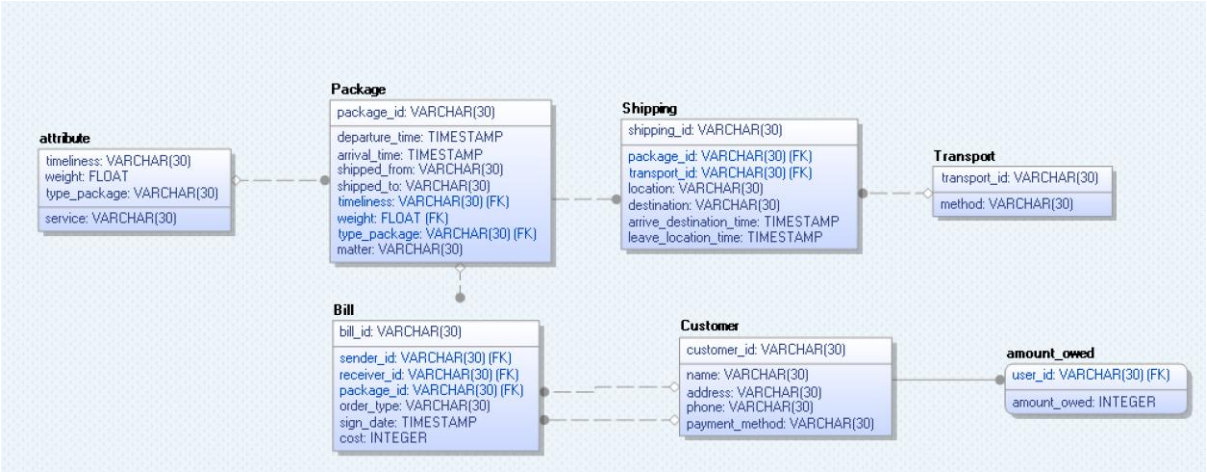


bill의 functional dependency는 {bill_id -> sender_id, receiver_id, package_id, order_type, sign_date, cost} 이고 bill_id는 bill entity set의 superkey이므로 bill은 bcnf form을 만족한다.

shipping의 functional dependency는 { shipping_id -> package_id, transport_id, location, destination, leave_location_time, arrive_destination_time} 이고 shipping_id는 superkey이므로 shipping은 bcnf form을 만족한다.

transport의 functional dependency는 {transport_id -> method} 이고 transport_id는 superkey이므로 bcnf form을 만족한다.

이러한 logical schema를 기반으로 physical schema를 생성하였다.



모든 id 값은 integer type의 값을 저장하도록 하였다.

장소에 관련된 속성인 (shipped_to, shipped_from, location, destination,address)은 varchar(30) type으로 설정하여 다양한 주소에 대한 정보를 저장할 수 있도록 하였다.

cost, amount_owed는 가격을 integer type의 값으로 저장하도록 하였다.

weight는 소수점까지 표현하기 위해 float type의 값을 저장하도록 하였다.

시간에 관련된 속성인 (departure_time, arrival_time, arrive_destination_time, leave_location_time, sign_date)는 timestamp type으로 값을 저장하도록 하였다.

method는 truck, plane, warehouse 등의 값을 저장하므로 varchar(30) type으로 설정하였다.

name은 데이터 설정은 영어 이름으로 설정했기 때문에 varchar(30)으로 넉넉하게 설정하였다.

phone은 숫자 중간에 -가 들어갈 수 있으므로 varchar(30)으로 설정하였다.

payment_method는 card, phone, internet, account 등의 값을 저장하므로 varchar(30)으로 설정하였다.

order_type은 purchase, refund들을 값으로 저장하므로 varchar(30)으로 설정하였다.

type_package는 small_box, flat_envelope, large_box의 값을 저장하므로 varchar(30)으로 설정

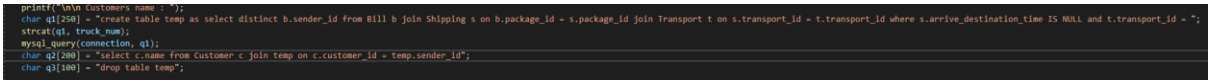
하였다.

timeliness는 overnight, second_time, longer의 값을 저장하므로 varchar(30)으로 설정하였다. service는 premier, reasonable, normal 등의 값을 저장하므로 varchar(30)으로 설정하였다.

matter는 hazardous , international 등의 값을 저장하므로 varchar(30)으로 설정하였다.

C code implement

Query 1-1 : 특정 트럭이 충돌이 발생했다면, 그 시점에 트럭에 적재되어 있는 화물들을 보낸 고객들을 조회하는 쿼리이다.




```
create table temp as select distinct b.sender_id from Bill b join Shipping s on
b.package_id = s.package_id join Transport t on s.transport_id = t.transport_id where
s.arrive_destination_time IS NULL and t.transport_id = (some id)
```

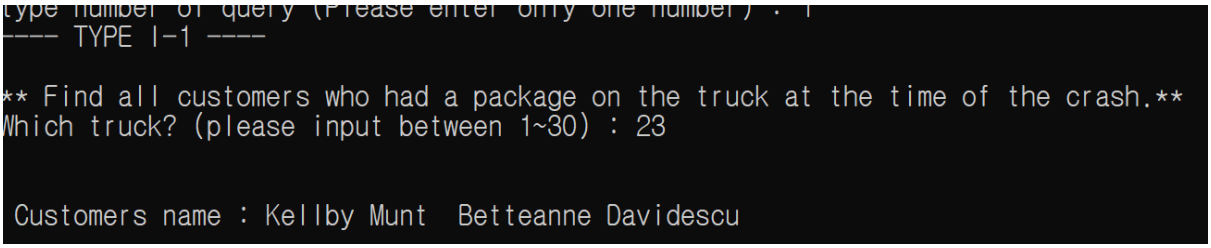
bill 과 shipping을 join하여 해당 주문에 대한 모든 배송 현황 중에 도착시간이 NULL인 값들, 즉 충돌을 가정했을 때 현재 배송중이던 물품들을 가져오고 다시 transport와 join하여 조회하고자 하는 트럭에 대한 배송 현황을 전부 임시 테이블에 저장하고 보낸 고객의 id만 select 하였다.

```
select c.name from Customer c join temp on c.customer_id = temp.sender_id
```

그 후 customer와 temp를 join하여 해당하는 이름만 추출하였다.

작업이 끝나고 나면 생성한 임시 테이블을 제거해주었다.

```
drop table temp
```



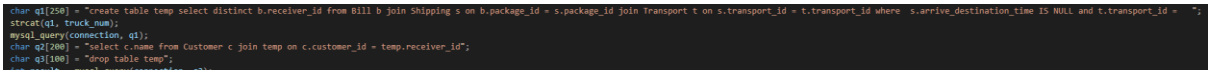
database에서 같은 쿼리를 실행시킨 결과이다.

	sender_id	transport_id	arrive_destination_time
▶	2	23	NULL
	10	23	NULL

	customer_id	name
▶	2	Kellby Munt
	10	Betteanne Davidescu

transport_id가 23이고 arrive_destination_time이 null인 sender들만 조회되고 있음을 알 수 있다.

Query 1-2 : 특정 트럭에 충돌이 발생하였다면, 그 시점에 트럭에 적재되어 있는 화물들을 받을 고객들을 조회하는 쿼리이다.




```
create table temp select distinct b.receiver_id from Bill b join Shipping s on
b.package_id = s.package_id join Transport t on s.transport_id = t.transport_id where
s.arrive_destination_time IS NULL and t.transport_id = (some id)
```

1-1과 마찬가지로 bill 과 shipping을 join하여 해당 주문에 대한 모든 배송 현황 중에 도착시간이 NULL인 값들,즉 충돌을 가정했을 때 현재 배송중이던 물품들을 가져오고 다시 transport와 join하여 조회하고자 하는 트럭에 대한 배송 현황을 전부 임시 테이블에 저장하고 받을 예정인 고객의 id만 select하였다.

```
select c.name from Customer c join temp on c.customer_id = temp.receiver_id
```

그 후 customer와 temp를 join하여 해당하는 이름만 추출하였다. 1-1과 다른 점은 receiver의 id를 사용한다는 것이다.

작업이 끝나고 나면 생성한 임시 테이블을 제거해주었다.

```
drop table temp
```

```
---- TYPE I-2 ----
** Find all recipients who had a package on that truck at the time of the crash.**
Which truck? (please input between 1~30) : 23


Recipient Name : Marnia McIlvoray       Renata Jzak
```

실제로 db에서 같은 쿼리를 실행해본 결과,

	receiver_id	transport_id	arrive_destination_time
▶	4	23	NULL
	3	23	NULL

	customer_id	name
▶	4	Marnia McIlvoray
	3	Renata Jzak

transport_id가 23이고 arrive_destination_time이 null인 receiver들만 조회되고 있음을 알 수 있다.

```
Which truck? (please input between 1~30) : 2

Recipient Name :
```

만약 충돌난 트럭에 아무런 화물이 없었다면 아무 고객의 이름도 출력되지 않는다.

QUERY 1-3 : 특정 트럭이 충돌이 발생했다면 그 전에 가장 최근 배송된 화물에 대한 정보를 조회하는 쿼리이다.


```
CREATE TABLE crushed_package AS SELECT p.package_id, p.departure_time, p.arrival_time FROM Package p JOIN Shipping s ON p.package_id = s.package_id JOIN Transport t ON s.transport_id = t.transport_id WHERE t.transport_id = some id
```

crushed_package라는 임시 테이블을 생성하였다. 해당 테이블은 특정 트럭이 배송했던 모든 화물을 저장한다. package와 shipping을 package_id로 join하고 다시 transport_id로 특정 트럭에 대해 조회할 수 있다.

```
SELECT c.package_id, c.departure_time, c.arrival_time FROM crushed_package c WHERE c.arrival_time IS NOT NULL ORDER BY c.arrival_time DESC LIMIT 1
```

그 다음 임시 테이블에서 도착시간을 내림차순으로 정렬하고 맨 위의 있는 것이 가장 최근의 물품이기 때문에 1개의 열만 저장하게 조건을 처리한다. 단 NULL인 값들은 배송이 실패한 물품이기 때문에 제외한다. 그리고 배송에 실패한 시점을 현재 시간으로 하기로 가정한다.

```
Which truck? (please input between 1~30) : 23

Successful Delivery :
```

다음과 같이 그런 값이 없는 트럭은 아무런 정보도 출력되지 않지만

```
---- TYPE I-3 ----
** Find the last successful delivery by that truck prior to the crash.**
Which truck? (please input between 1~30) : 11


Successful Delivery :
 package id : 45              departure time : 2022-10-31 12:31:00          arrival time : 2022-11-02 18:31:00
```

충돌 이전에 성공적으로 배송된 물품이 존재한다면 위와 같이 출력되는 것을 확인할 수 있다.

```
CREATE TABLE crushed_package AS SELECT p.package_id, t.transport_id , p.arrival_time FROM Package p JOIN Shipping s ON p.package_id = s.package_id JOIN Transport t ON s.transport_id = t.transport_id
SELECT c.package_id, c.transport_id, c.arrival_time FROM crushed_package c WHERE c.arrival_time IS NOT NULL;
drop table crushed_package;
```

db에서 같은 쿼리를 실행시킨 결과이다.

가장 최근의 결과를 출력하는 것이 맞는 지 모든 값을 출력하도록 하였다.

	package_id	transport_id	arrival_time
▶	9	11	2022-06-20 20:05:00
	45	11	2022-11-02 18:31:00

충돌 이전에 존재하는 배송은 2개이며 그 중 가장 최근의 배송인 45번 package를 출력하고 있음을 알 수 있다.

QUERY 2 : 특정 연도에 가장 많은 배송을 보낸 고객을 조회하는 쿼리이다.

```
if (!strcmp(year, "0")) break;
printf("\n\nCustomer Name and number of packages : ");
char q1[1000] = "create table temp as select sender_id , count(*) cnt from Bill where Year(sign_date) = ";
char q2[200] = " group by sender_id ORDER BY cnt DESC LIMIT 1";
char q3[200] = "select c.name , temp.cnt from Customer c join temp on temp.sender_id = c.customer_id";
char q4[100] = "drop table temp;";
strcat(q1, year);
```

```
create table temp as select sender_id , count(*) cnt from Bill where Year(sign_date) = (some year) group by sender_id ORDER BY cnt DESC LIMIT 1
```

임시 테이블을 생성하고 Bill에서 주문 연도와 특정 연도와 일치하는 주문들의 합을 group by sender_id를 통해 sender_id에 따라 count로 모두 구해주고 order by를 통해 내림차순으로 정렬하여 저장하였다.

이러한 쿼리의 결과로 temp에는 가장 값이 큰 주문량을 가진 sender_id 하나만 저장될 것이다.

```
select c.name , temp.cnt from Customer c join temp on temp.sender_id = c.customer_id
```

그 다음 temp에 저장된 sender_id에 해당하는 custom의 이름과 총 주문량을 select하여 출력하였다.

```
---- TYPE II ----
** Find the customer who has shipped the most packages in certain year**
 Which Year? (2022 or 2023) : 2022


Customer Name and number of packages :  Sophie Corby 7
```

위 처럼 입력한 연도의 가장 많은 배송을 보낸 고객의 이름과 총 배송량을 조회할 수 있다.

```
create table temp as select sender_id , count(*) cnt from Bill where Year(sign_date) = 2022 group by sender_id ;
select c.name , temp.cnt from Customer c join temp on temp.sender_id = c.customer_id;
drop table temp
```

실제 db에서 같은 쿼리를 실행시킨 결과,

	name	cnt
▶	Sophie Corby	7
	Cortie Patshull	5
	Faulkner McKiernan	5
	Hillery O'Dunniom	4
	Josepha Garralts	4
	Kellby Munt	3
	Renata Jzak	3
	Marnia McIlvoray	3
	Noble Cherry Holme	3
	Tandie Curnnokk	3
	Enrico Marchent	3

가장 큰 값인 고객의 이름을 출력하고 있음을 알 수 있다.

QUERY 3 : 특정 연도에 가장 많은 금액을 지불한 고객의 이름을 조회하는 쿼리이다.

```
printf("\n\nCustomer Name and total payment amount : ");
char q1[1000] = "create table temp as select sender_id, sum(cost) pay from Bill where year(sign_date) =";
char q2[100] = " group by sender_id";
char q3[200] = "SELECT c.name , t.pay FROM Customer c JOIN temp t ON c.customer_id = t.sender_id WHERE t.pay = (SELECT MAX(pay) FROM temp)";
char q4[100] = "drop table temp;";
```

```
create table temp as select sender_id, sum(cost) pay from Bill where year(sign_date) =
some year group by sender_id
```

임시 테이블인 temp를 생성하고 bill에서 입력한 특정 연도에 해당하는 cost들의 합을 pay로 명명하고 sender_id에 따라 계산하여 저장한다.

```
SELECT c.name , t.pay FROM Customer c JOIN temp t ON c.customer_id = t.sender_id WHERE
t.pay = (SELECT MAX(pay) FROM temp)
```

temp에서 pay들의 최대값을 t.pay에 저장하고 sender_id에 해당하는 customer의 이름과 같이 저장하여 출력한다.

```
--- TYPE III -----
** Find the customer who has spent the most money on shipping in the certain year **
 Which Year? (2022 or 2023) : 2022


Customer Name and total payment amount : Sophie Corby 347457
```

위와 같이 해당 연도에 가장 많은 지출을 한 고객의 이름과 지출의 합을 출력하는 것을 볼 수 있다.

```
create table temp as select sender_id, sum(cost) pay from Bill where year(sign_date) = 2022 group by sender_id;
SELECT c.name , t.pay FROM Customer c JOIN temp t ON c.customer_id = t.sender_id ;
drop table temp
```

db에서 실행시킨 쿼리이다. 모든 총합을 출력해 최대값이 맞는 지 확인하기 위해 max를 where에서 없애고 실행시켰다.

name	pay
Sophie Corby	347457
Faulkner McKiernan	285638
Josepha Garralts	255669
Hillery O'Dunniom	230587
Cortie Patshull	225633
Kellby Munt	194339
Renata Jzak	181737
Noble Cherry Holme	177860
Marnia McIlvoray	175800
Enrico Marchent	159646
Tandie Curnnokk	137897

최대값을 가지는 고객의 이름을 출력함을 알 수 있다.

QUERY 4 : 배송되기로 한 기간내에 배송되지 않은 배송 물품들을 전부 조회하는 쿼리이다.

```
char q2[200] = "create table t2 as select package_id from Package where timeliness = 'overnight' and departure_time + INTERVAL 1 DAY  < arrival_time";
char q3[200] = "create table t3 as select package_id from Package where timeliness = 'second_day' and departure_time + INTERVAL 2 DAY  < arrival_time";
char q4[200] = "create table t4 as select package_id from Package where timeliness = 'longer' and departure_time + INTERVAL 5 DAY  < arrival_time";
char q5[300] = "SELECT package_id FROM t2 UNION SELECT package_id FROM t3 UNION SELECT package_id FROM t4 ORDER BY package_id ASC";
char q6[100] = "drop table t2,t3,t4";
```

```
create table t2 as select package_id from Package where timeliness = 'overnight' and
departure_time + INTERVAL 1 DAY  < arrival_time
create table t3 as select package_id from Package where timeliness = 'second_day' and
departure_time + INTERVAL 2 DAY  < arrival_time
create table t4 as select package_id from Package where timeliness = 'longer' and
departure_time + INTERVAL 5 DAY  < arrival_time
```

위 세 쿼리는 임시 테이블을 생성하고 각 예정 배송 기간보다 실제 배송완료 기간이 더 오래 걸린 물품들을 저장한다.

예정 배송 기간이 'overnight' , 'second_day', 'longer'라면 각각 하루, 이틀, 5일 까지의 유예 기간이 주어지는 것으로 설정하였다.

```
SELECT package_id FROM t2 UNION SELECT package_id FROM t3 UNION SELECT package_id FROM
```

```
t4 ORDER BY package_id ASC
```

그 다음 임시테이블들을 모두 union으로 저장하여 package_id들을 모두 출력하였다. 가독성을 위해 package_id를 오름차순으로 정렬하여 출력하였다.

```
--- TYPE IV ----
** Find those packages that were not delivered within the promised time.**
1 17 34 45 47 51 64 65 73 80 81 88 91 95 98
```

```
create table t2 as select * from Package where timeliness = 'overnight' and departure_time + INTERVAL 1 DAY  < arrival_time;
create table t3 as select * from Package where timeliness = 'second_day' and departure_time + INTERVAL 2 DAY  < arrival_time;
create table t4 as select * from Package where timeliness = 'longer' and departure_time + INTERVAL 5 DAY  < arrival_time;
SELECT package_id, timeliness, departure_time,arrival_time FROM t2 UNION SELECT package_id, timeliness, departure_time,arrival_time FROM t3 UNION SELECT package_id, timeliness, departure_time,arriva
drop table t2,t3,t4
```

db에서 실행시켜본 결과,

package_id	timeliness	departure_time	arrival_time
1	second_day	2023-02-01 23:21:00	2023-02-04 03:21:00
17	overnight	2023-01-11 13:17:00	2023-01-13 00:17:00
34	overnight	2023-01-03 07:31:00	2023-01-04 16:31:00
45	second_day	2022-10-31 12:31:00	2022-11-02 18:31:00
47	overnight	2023-01-09 19:34:00	2023-01-11 04:34:00
51	overnight	2022-11-29 02:42:00	2022-11-30 13:42:00
64	overnight	2023-01-24 07:57:00	2023-01-25 17:57:00
65	second_day	2023-02-10 15:57:00	2023-02-12 17:57:00
73	overnight	2022-07-12 04:45:00	2022-07-13 08:45:00
80	second_day	2023-04-08 00:41:00	2023-04-10 11:41:00
81	overnight	2022-10-12 19:16:00	2022-10-14 05:16:00
88	overnight	2022-06-28 22:14:00	2022-06-30 01:14:00

1번의 경우를 살펴보면 timeliness가 second_day 이므로 2일 내에 배송이 완료되어야 한다.

하지만 출발 시간과 도착 시간이 23-02-01 -> 23-02-04 이므로 대략 3일이 걸린 것을 알 수 있다. 즉 실제 배송 시간보다 오래 걸린 것을 알 수 있다.

Query 5 : 모든 고객에 대해 특정 연월에 결제한 명세서를 출력하는 쿼리이다.

```
char q1[200] = "select c.name , a.amount, c.payment_method from Customer c join amount_owed a on c.customer_id = a.user_id";
state = mysql_query(connection, q1);
if (state == 0) {
    sql_result = mysql_store_result(connection);
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
        strcpy(names[index], sql_row[0]);
        strcpy(amount[index], sql_row[1]);
        strcpy(method[index], sql_row[2]);
        index++;
    }
    mysql_free_result(sql_result);
}
```

```
char input[10];
char month[10];
char year[10];
```

먼저 select 함수로 customer에 존재하는 모든 고객의 이름과 미지불 금액, 결제 방식을 char

배열에 저장한다.

이렇게 구현한 이유는 sql_store_result는 현재 작동중인 하나의 쿼리에 대한 결과만 출력하므로 동시의 두개의 쿼리를 실행시키면 error가 발생하기 때문이다.

즉 이름과 같은 개인정보를 조회하면서 동시에 명세서를 조회하는 쿼리를 실행시킬 수 없기 때문에 이러한 방식을 사용하였다.

```
for (int i = 0; i < index; i++) {
    printf("| %-30s | %-15s | %-15s |\n", "Customer name", "Amount owed", "Payment method");
    printf("-------------------------------------------------------------------------\n");
    printf("| %-30s | %-15s | %-15s |\n\n", names[i], amount[i], method[i]);

    printf("| %-20s | %-10s | %-15s | %-15s | %-20s |\n", "Package Number", "Cost", "Order Type", "Payment Method", "Order Date");
    printf("-------------------------------------------------------------------------\n");
    char q2[300] = "select b.package_id , b.cost , b.order_type, c.payment_method, b.sign_date from Bill b join Customer c on b.sender_id = c.customer_id where c.name = ";
    strcat(q2, "\"");
    strcat(q2, names[i]);
    strcat(q2, "\"");
    strcat(q2, " and month(b.sign_date) = ");
    strcat(q2, month);
    strcat(q2, " and year(b.sign_date) = ");

    strcat(q2, year);

    state = mysql_query(connection, q2);
    if (state == 0) {
        sql_result = mysql_store_result(connection);
        while ((sql_row = mysql_fetch_row(sql_result)) != NULL) {
            printf("| %-20s | %-10s | %-15s | %-15s | %-20s |\n", sql_row[0], sql_row[1], sql_row[2], sql_row[3], sql_row[4]);
        }
        mysql_free_result(sql_result);
    }
    printf("\n");
```

c에서 지원하는 문자열 서식 기능을 통해 보기 좋은 명세서를 출력하도록 했다.

쿼리문에서는 미리 입력 받은 names[i]에 해당하는 고객을 찾기 위해 customer에서 name과 일치하는 고객을 찾고 bill의 sender_id와 특정 고객의 customer_id 가 같아야 하며, 미리 입력받은 year과 month가 위에서 조회한 bill의 sign_date과 일치하는 데이터들만 출력하도록 설정하였다.

```
| Customer name                  | Amount owed     | Payment method  |
-------------------------------------------------------------------------
| Noble Cherry Holme             | 0               | internet        |

| Package Number       | Cost       | Order Type      | Payment Method  | Order Date          |
-------------------------------------------------------------------------
| 5                    | 60752      | purchase        | internet        | 2022-06-07 04:56:32 |
| 35                   | 98829      | refund          | internet        | 2022-06-27 12:33:07 |
```

2022-06을 입력값으로 조회한 결과이다. 현재 이 고객은 6월에 2개의 배송을 주문했으며 해당 주문에 대한 다양한 정보가 조회되는 것을 알 수 있다.

```
| Customer name                  | Amount owed     | Payment method  |
-------------------------------------------------------------------------
| Marnia McIlvoray               | 0               | internet        |

| Package Number       | Cost       | Order Type      | Payment Method  | Order Date          |
-------------------------------------------------------------------------

| Customer name                  | Amount owed     | Payment method  |
-------------------------------------------------------------------------
```

다음과 같이 아무것도 6월에 주문하지 않은 고객은 아무런 정보도 출력되지 않는 것을 확인할 수 있다.

QUERY 0 : 0을 입력하면 반드시 현재 쿼리 선택창에서 뒤로 나가도록 한다.

```
---- TYPE I-1 ----

** Find all customers who had a package on the truck at the time of the crash.**
Which truck? (please input between 1~30) : 20


 Customers name : Sophie Corby  Cortie Patshull  Marnia McIlvoray  Enrico Marchent

Which truck? (please input between 1~30) : 30


 Customers name : Tandie Curnnokk  Hillery O'Dunniom  Faulkner McKiernan  Kellby Munt  Renata Jzak

Which truck? (please input between 1~30) : 0

----- Subtypes int TYPE I -----
        1. TYPE I-1.
        2. TYPE I-2.
        3. TYPE I-3.
type number of query (Please enter only one number) :
```

어떤 쿼리에서 조회 작업을 입력값을 주면서 실행시키면 종료되는 것이 아니라 다시 입력값을 받아 반복적으로 값을 조회할 수 있도록 구현하였다.

작업을 종료하고 싶다면 0을 입력하여 이전 쿼리 선택창으로 넘어가게 된다.

위에서 query 1-1의 조회 작업을 처리하고 있다가 0을 입력하자 sub query를 선택하는 창으로 돌아간 것을 볼 수 있다.

```
----- Subtypes int TYPE I -----
        1. TYPE I-1.
        2. TYPE I-2.
        3. TYPE I-3.
type number of query (Please enter only one number) : 0

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

        1. TYPE I
        2. TYPE II
        3. TYPE III
        4. TYPE IV
        5. TYPE V
        0. QUIT
type number of query (Please enter only one number) :
```

또한 TYPE1에서 sub query들을 선택하는 창에서도 0을 입력해야만 main query를 선택하는 창으로 돌아가도록 구현하였다.

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

        1. TYPE I
        2. TYPE II
        3. TYPE III
        4. TYPE IV
        5. TYPE V
        0. QUIT
type number of query (Please enter only one number) : 0

C:₩Users₩임형준₩source₩repos₩db pj2₩x64₩Debug₩db pj2.exe(프로세스 11600개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

마지막으로 main query 선택창에서도 0을 입력한다면 완전히 프로그램을 종료하도록 설정하였다.

```
char clear[100] = "drop table Customer, Package, Transport, Shipping, attribute, Bill, amount_owed";
mysql_query(connection, clear);
mysql_close(connection);
return 0;
```

그리고 ctrl + c로 강제 종료하는 것이 아니라 정상적인 경로 즉 메인 쿼리 선택창에서 0으로 종료하게 된다면 db에 생성되어 있는 모든 table들을 삭제하도록 구현하였다.