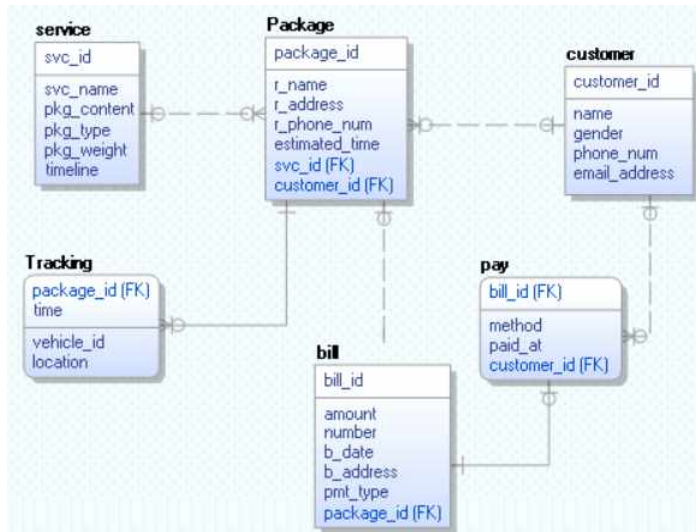
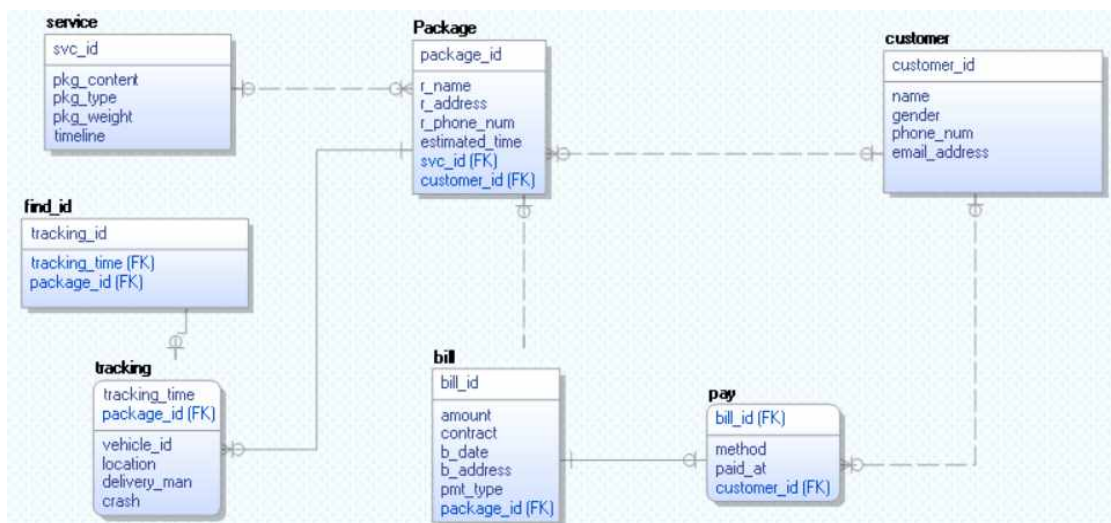


**CSE4110 - Database System**  
**Project2. Normalization and Query Processing**

이름 : 김희진  
학년 : 3  
학번 : 20211530  
전공 : 컴퓨터공학과



기존(project 1) schema



project 2 logical schema

## 0. project1에서 추가한 점

아래에서 설명할 BCNF decomposition를 제외하고 달라진 점은 **find\_id** entity에 `tracking_id`, `tracking` entity에 `crash`와 배송기사를 나타내는 `delivery_man`을 추가한 것입니다. `delivery_man` property는 이번 project의 query에는 사용되지 않지만 project1의 요구사항 중 있었지만 제대로 수행하지 못하여 추가하였습니다. 또한, `Tracking_id`는 query를 구현하던 중 필요하다고 판단되어 추가한 property입니다. `tracking_id`는 각 package가 tracking될 때, package와 시간별로 unique한 값을 가지는 속성입니다. 이것이 있어야 tracking이 더 용이하게 검색될 수 있어 추가하였습니다. `crash` property도 query를 수행함에 있어 필요하다고 판단되어 넣어준 속성입니다. 그리고 `bill` entity의 이전의 `number` property는 이름이 모호하여 `contract`로 변경하여 주었습니다. 또한, `service` entity의 `svc_name`은 `svc_id`와 같

은 역할을 수행하고 있었기에 삭제해 주었습니다.

## 1. BCNF decomposition

BCNF simplified test를 통해 해당 entity가 BCNF를 충족하는지 확인하였고 충족하지 않는 entity는 decomposition 해주었습니다. 이를 위해 각 entity의 fd를 살펴보고 a->b에서 a가 superkey에 해당하지 않고 trivial하지 않은 경우 decomposition을 진행하였습니다.

### ① Tracking entity

유일하게 decomposition을 진행한 entity 입니다. 앞서 말한 tracking\_id를 추가한 entity를 고려했을 때 fd는 다음과 같습니다.

tracking\_id -> package\_id, tracking\_time, delivery\_man, vehicle\_id, location, crash  
package\_id, tracking\_time -> delivery\_man, vehicle\_id, location, crash

두 번째 fd에서 crash는 time을 통해 판별되고 delivery\_man, vehicle\_id, location은 package\_id와 tracking\_time이 같을 때 항상 같으므로 이와 같이 판단해 주었습니다.

첫 번째 fd의 좌측항 tracking\_id는 primary key이므로 super key입니다. 그러므로 해당 fd는 BCNF를 위반하지 않습니다. 두번째 fd의 좌측 항은 super key가 아니므로 BCNF 위반하여 decomposition을 진행해 주어야 합니다. BCNF를 위반할 때, a->b에 대해 (a와 b의 합집합), (R-(b-a)) 형태로 decomposition 해야하기에 R1(package\_id, tracking\_time, delivery\_man, vehicle\_id, location), R2(tracking\_id, package\_id, tracking\_time)으로 decomposition 해주었습니다. 우선 R1에 대해서 package\_id, tracking\_time이 pk가 되므로 BCNF를 충족합니다. 또한, R2에 대해서, fd tracking\_id->package\_id, tracking\_time이 존재하는데 tracking\_id는 pk이므로 BCNF를 충족합니다.

R1으로 분해한 entity의 이름을 tracking, R2로 분해한 entity의 이름을 find\_id로 이름지었고 앞서 simplified test의 결과로 보듯 이는 모두 BCNF decomposition을 충족합니다.

### ② service entity

이 entity의 fd는 다음과 같습니다.

svc\_id -> svc\_name, pkg\_weight, pkg\_type, pkg\_content, timeline

svc\_name, pkg\_weight, pkg\_type, pkg\_content, timeline -> svc\_id

첫 번째 fd에서 좌측항 svc\_id는 pk이므로 super key에 해당합니다. 그러므로 첫 번째 fd는 BCNF를 위반하지 않습니다. 두 번째 fd에서 좌측항의 closure를 구하면 (svc\_name, pkg\_weight, pkg\_type, pkg\_content, timeline) -> (svc\_name, pkg\_weight, pkg\_type, pkg\_content, timeline, svc\_id) 입니다. 즉, 두 번째 fd의 좌측항은 superkey입니다. 그러므로 두 번째 fd도 BCNF를 위반하지 않으므로 해당 entity는 decomposition을 진행해주지 않았습니다.

### ③ package entity

이 entity의 fd는 다음과 같습니다.

package\_id -> r\_name, r\_address, r\_phone\_num, estimated\_time, svc\_id, customer\_id

svc\_id -> r\_name, r\_address, r\_phone\_num, estimated\_time, package\_id, customer\_id

즉, fd의 좌측항이 primary key, super key인 fd만 존재합니다. 그러므로 해당 entity는 BCNF를 위반하지 않는 형태이므로 decomposition을 진행히 주지 않았습니다.

### ④ customer entity

이 entity의 fd는 다음과 같습니다.

customer\_id -> name, gender, phone\_num, email\_address

즉, fd의 좌측항이 primary key인 fd만 존재합니다. 그러므로 해당 entity는 BCNF를 위반하지 않는 형태이므로 decomposition을 진행히 주지 않았습니다.

### ⑤ pay entity

이 entity의 fd는 다음과 같습니다.

bill\_id->method, paid\_at, customer\_id

즉, fd의 좌측항이 primary key인 fd만 존재합니다. 그러므로 해당 entity는 BCNF를 위반하지 않는 형태이므로 decomposition을 진행히 주지 않았습니다.

## ⑥ bill entity

이 entity의 fd는 다음과 같습니다.

bill\_id -> amount, contract, b\_date, b\_address, pmt\_type, package\_id

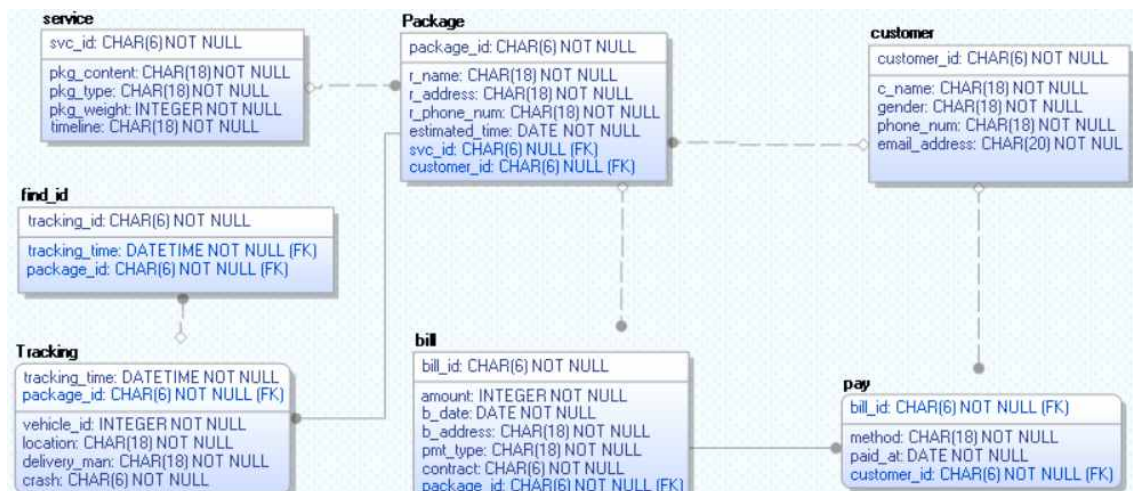
package\_id -> amount, contract, b\_date, b\_address, pmt\_type, bill\_id

즉, fd의 좌측항이 primary key, super key인 fd만 존재합니다. 그러므로 해당 entity는 BCNF를 위반하지 않는 형태이므로 decomposition을 진행해 주지 않았습니다.

## 2. Physical schema

앞선 logical schema를 바탕으로 도출한 physical schema는 다음과 같습니다.

각 entity와 relation에 대해 data type, relation type, null 여부 등을 지정했습니다.



### ① package

- package\_id : “P00000”와 같은 형태로 저장되므로 data type은 char(6)으로 정해주었습니다. primary key 이므로 자동으로 not null입니다.

- r\_name : 수령인의 이름을 담고 data type은 char(18)로 지정해 주었습니다. not null로 설정해주어 수령인의 이름이 꼭 들어가게 해주었습니다.

- r\_address : 수령인의 주소를 담고 data type은 char(18)로 지정해 주었습니다. 실제라면 더 길게 지정해야겠지만 해당 프로젝트에서 주소는 “~구” or “~동” 형태로 지정할 것이기에 길이를 18로 지정했습니다. 수령인의 주소는 꼭 필요한 정보이므로 not null로 설정했습니다

다.

- r\_phone\_num : 수령인의 휴대폰 번호를 "010-0000-0000"형태로 담고 data type은 char(18)로 지정해주었습니다. 이도 수령을 위해 꼭 필요한 정보이므로 not null로 설정했습니다.
- estimated\_time : 도착 예정시간으로 data type은 date입니다. 년부터 초까지 설정가능하지만 "yyyy-mm-dd hh:mm"형태로 분까지만 지정했습니다. query 계산을 위해 꼭 필요한 정보이므로 not null로 설정해주었습니다.

## ② customer

- customer-id : "C00000"와 같은 형태로 저장되므로 data type은 char(6)으로 정해주었습니다. primary key이므로 자동으로 not null입니다.
- c\_name : 고객의 이름을 담고 data type은 char(18)입니다. 고객에게 청구서를 청구할 때 이름은 필요한 정보이므로 not null로 설정했습니다.
- gender : 고객의 성별을 담고 여자는 "f", 남자는 "m"로 저장됩니다. data type은 넉넉히 char(18)로 지정해 주었습니다. not null로 설정하여 성별이 꼭 들어가게 설정해 주었습니다.
- phone\_num : 고객의 휴대폰 번호가 "010-0000-0000"과 같은 형태로 저장되므로 char(18)로 설정해 주었습니다. 고객의 연락처는 문자배송등을 위해 필요하므로 not null로 설정했습니다.
- email\_address : 고객의 이메일 주소를 담으며 data type은 char(20) 으로 설정했습니다. 휴대폰 연락이 되지 않을 때 연락을 위한 수단이므로 not null로 설정하여 꼭 입력받도록 하였습니다.

## ③ pay

- method : 지불 방법으로 "credit\_card", "account\_trasfer"등의 정보가 저장되므로 data type은 char(18)로 설정했습니다. 이는 꼭 필요한 정보이므로 not null로 설정했습니다.
- piad\_ad : 지불 시기로 data type은 date입니다. 년부터 초까지 설정가능하지만

“yyyy-mm-dd hh:mm”형태로 분까지만 지정했습니다. 이도 꼭 필요한 정보이므로 not null로 설정했습니다.

#### ④ bill

- bill\_id : “B00000”와 같은 형태로 저장되므로 data type은 char(6)으로 정해주었습니다. primary key이므로 자동으로 not null입니다.
- amount : 지불할 가격이므로 data type은 integer입니다. 꼭 필요한 정보이므로 not null로 설정하였습니다.
- b\_date : 청구서가 생성된 날짜로 data type은 date입니다. 년부터 초까지 설정가능하지만 “yyyy-mm-dd hh:mm”형태로 분까지만 지정했습니다. 이도 query생성에 꼭 필요한 정보이므로 not null로 설정했습니다.
- b\_address : 청구서가 보내질 주소로 data type은 char(18)입니다. 실제라면 더 길게 지정해야겠지만 해당 프로젝트에서 주소는 “~구” or “~동” 형태로 지정할 것이기에 길이를 18로 지정했습니다. 청구서가 보내질 주소는 꼭 필요한 정보이므로 not null로 설정했습니다.
- pmt\_type : “prepaid” or “postpaid”가 저장될 property로 data type은 char(18)입니다. 이는 꼭 필요한 정보이므로 not null로 설정했습니다.
- contract : 계약 여부를 나타내는 property로 “yes” or “no”가 저장되므로 data type은 char(6)으로 설정했습니다. 이도 꼭 필요한 정보이므로 not null로 설정했습니다.

#### ⑤ tracking

- tracking\_time : 추적시간을 나타내는 property로 data type은 date입니다. 년부터 초까지 설정가능하지만 “yyyy-mm-dd hh:mm”형태로 분까지만 지정했습니다. primary key이므로 not null로 자동 설정 됩니다.
- vehicle\_id : 숫자로 구성되는 id로 data type은 integer 입니다. 이는 query에 꼭 필요한 정보이므로 not null로 설정해줍니다.
- location : 추적하는 장소로 data type은 char(18) 입니다. 이는 꼭 필요한 정보이므로 not

null로 설정해 줍니다.

- delivery\_man : 해당 vehicle의 배달 기사로 data type은 char(18) 입니다. 이도 not null로 설정해주어 정보가 꼭 들어가도록 했습니다.
- crash : 해당 vehicle가 해당 시간에 사고가 났는지 여부를 저장하며 “yes” or “no”의 값을 가집니다. data type은 char(6)이며 query 수행에 필요한 속성이므로 not null입니다.

#### ⑥ find\_id

- tracking\_id : “T00000”와 같은 형태로 저장되므로 data type은 char(6)으로 정해주었습니다. primary key이므로 자동으로 not null입니다.

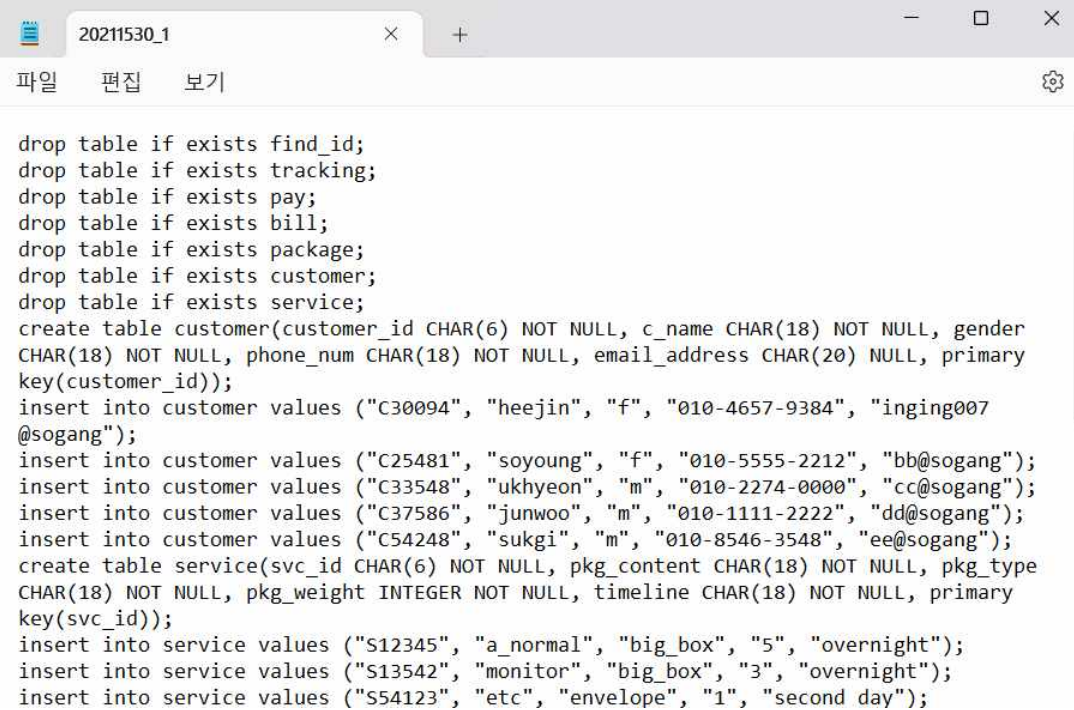
#### ⑦ service

- svc\_id : “S00000”와 같은 형태로 저장되므로 data type은 char(6)으로 정해주었습니다. primary key이므로 자동으로 not null입니다.
- pkg\_content : package 내용물이 저장되며 해외배송과 위험 물품에 대한 정보도 함께 담습니다. 해외배송은 a\_가 위험물품은 h\_가 내용물 앞에 추가로 적히게 됩니다. data type은 char(18)이며 service를 결정하는 요인이므로 not null로 설정해 정보가 꼭 들어가게 했습니다.
- pkg\_type : “big\_box”, “medium\_box”, “small\_box”, “envelope”의 type이 저장되고 data type은 char(18)로 설정해주었습니다. service를 결정하는 요인이므로 not null로 설정해 정보가 꼭 들어가게 했습니다.
- pkg\_weight : 물품의 무게로 data type은 integer입니다. 1보다 작은 값은 0으로 저장됩니다. service를 결정하는 요인이므로 not null로 설정해 정보가 꼭 들어가게 했습니다.
- timeline : “overnight”, “second\_day”, “longer”의 timeline이 저장되며 data type은 char(18)로 정했습니다. service를 결정하는 요인이므로 not null로 설정해 정보가 꼭 들어가게 했습니다.

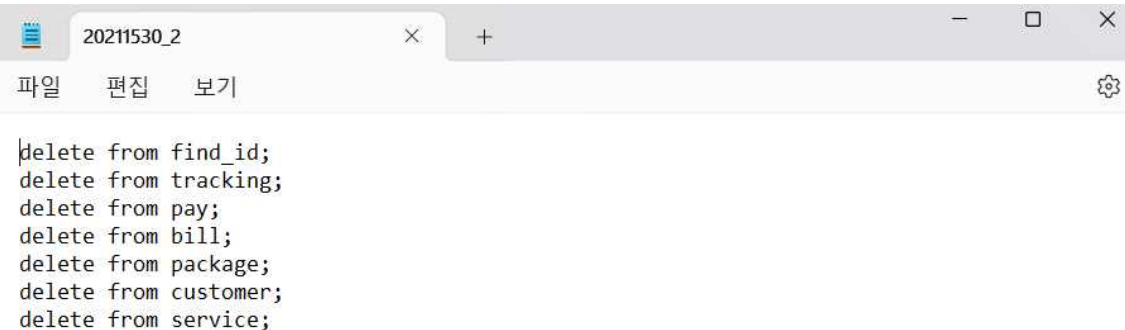


### 3. ODBC implementation within MySQL

#### ① crud queries를 위한 text file 생성



```
drop table if exists find_id;
drop table if exists tracking;
drop table if exists pay;
drop table if exists bill;
drop table if exists package;
drop table if exists customer;
drop table if exists service;
create table customer(customer_id CHAR(6) NOT NULL, c_name CHAR(18) NOT NULL, gender
CHAR(18) NOT NULL, phone_num CHAR(18) NOT NULL, email_address CHAR(20) NULL, primary
key(customer_id));
insert into customer values ("C30094", "heejin", "f", "010-4657-9384", "inging007
@sogang");
insert into customer values ("C25481", "soyoung", "f", "010-5555-2212", "bb@sogang");
insert into customer values ("C33548", "ukhyeon", "m", "010-2274-0000", "cc@sogang");
insert into customer values ("C37586", "junwoo", "m", "010-1111-2222", "dd@sogang");
insert into customer values ("C54248", "sukgi", "m", "010-8546-3548", "ee@sogang");
create table service(svc_id CHAR(6) NOT NULL, pkg_content CHAR(18) NOT NULL, pkg_type
CHAR(18) NOT NULL, pkg_weight INTEGER NOT NULL, timeline CHAR(18) NOT NULL, primary
key(svc_id));
insert into service values ("S12345", "a_normal", "big_box", "5", "overnight");
insert into service values ("S13542", "monitor", "big_box", "3", "overnight");
insert into service values ("S54123", "etc", "envelope", "1", "second day");
```



```
delete from find_id;
delete from tracking;
delete from pay;
delete from bill;
delete from package;
delete from customer;
delete from service;
```

create, insert, drop의 queries를 담고있는 20211530\_1.txt와 delete query를 담고있는 20211530\_2.txt를 생성했습니다. create는 table을 생성하며 insert는 생성된 create에 value을 넣어줍니다. 이때 project에서 요구하는 query가 수행됨을 확인할 수 있도록 데이터를 만들었습니다. drop은 query를 연결했을 때 이미 table이 존재하면 drop 시켜주는 역할을 수행하도록 쉘 앞부분에 입력했습니다. delete는 프로젝트에서 요구된 query가 모두 끝나고 수행될 query입니다.

#### ② 요구된 query에 대한 code 작성

```
Connection Succeed

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

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

connection에 성공하면 connection succeeded라는 문구가 뜬 후 선택할 수 있는 query types이 출력됩니다. 0을 제외한 나머지 쿼리에 대해 기술하겠습니다.

#### - Type 1

type 1을 선택하게되면 사고가 난 truck 번호를 입력받고 그 후 sub type을 고르게 됩니다. 다음 캡처화면은 17584 truck에 대해 1-1, 1-2, 1-3을 수행한 결과입니다.

해당 truck은 배송과정에서 두 개의 package를 담고 있는데 이 두 package의 customer는 같지만 recipient는 다릅니다. 하나의 package는 사고 전에 도착하였고 하나는 도착하지 못했습니다. 3개의 query 결과를 모두 확인할 수 있기에 해당 truck에 대해 query를 수행했습니다.

```
Select tye : 1

---- TYPE I ----
Enter the traffic ID you want to check : 17584

---- SUBTYPES IN TYPE I ----
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.

Select subtype : 1

---- TYPE I-1 ----
** Find all customer who had a package on the truck 17584 at the time of the crash **

customer_id : C30094
```

```
Select subtype : 2

---- TYPE I-2 ----
** Find all recipients who had a package on that truck 17584 at the time of the crash **

recipient name : heejin
```

```
Select subtype : 3
---- TYPE I-3 ----
** Find the last successful delivery by that truck prior to the crash **
last successful time : 2023-06-09 19:30:00
last successful delivery : P10006
```

1-1의 경우 사고가 난 시점 truck에 있던 package의 고객 id가 올바르게 출력된 모습입니다. 추가로, 사고가난 시점에 두 명의 고객에 의한 택배를 담고 있던 truck 9545에 대해 조회해본 결과 두 명의 고객을 올바르게 출력하는 모습을 확인할 수 있습니다.

```
---- TYPE I ----
Enter the traffic ID you want to check : 9545

---- SUBTYPES IN TYPE I ----
1. TYPE I-1.
2. TYPE I-2.
3. TYPE I-3.

Select subtype : 1
---- TYPE I-1 ----
** Find all customer who had a package on the truck 9545 at the time of the crash **

customer_id : C25481
customer_id : C54248
```

```
char query11[300] = "select distinct P.customer_id from package P, tracking T where
P.package_id=T.package_id and T.vehicle_id=";
```

```
strcat(query11, traffic); strcat(query11, " and T.crash='yes'");
```

해당 쿼리는 위와 같이 작성했으며 사고가 난 트럭에 한 명의 고객이 보낸 택배가 여러 개 들어있을 경우 한명의 고객만 출력하기 위해 distinct를 사용했습니다.

1-2의 경우 앞서 언급한 17584 트럭에 대해 트럭에 있던 package의 수령인은 두 명이지만 하나의 package는 사고가 나기 이전이므로 한 명만 출력되어야 합니다. 앞서 첨부한 출력화면에서 한 명의 수령인만 올바르게 출력됨을 확인할 수 있습니다.

```
char query12[300] = "select distinct P.r_name from Package as P, tracking as T where
P.package_id=T.package_id and T.vehicle_id=";
```

```
strcat(query12, traffic); strcat(query12, " and T.crash='yes'");
```

해당 쿼리는 위와 같이 작성했습니다.

1-3의 경우 사고가 나기 이전에 가장 마지막으로 배달에 성공한 날짜와 package\_id를 출력합니다. 앞서 첨부한 출력화면에서 올바르게 출력됨을 확인할 수 있습니다. 이를 위해 쿼리는 아

래와 같이 작성했습니다.

```
char query13[300] = "select distinct T1.tracking_time, T1.package_id from tracking T1, Package P
where T1.location=P.r_address and T1.tracking_time= (select max(T2.tracking_time) from tracking
T2 where T1.package_id=T2.package_id and vehicle_id=";
strcat(query13, traffic); strcat(query13, ");";
```

사고가 나기 이전에 가장 마지막으로 배달에 성공한 것을 고르기 위해 max(tracking\_time)과 tracking.location=package.r\_address 구문을 사용했습니다.

## - Type 2

```
----- TYPE II -----
** Find the customer who has shipped the most packages in certain year **

Which year? (yyyy format) : 2023

customer_id : C30094
customer_name : heejin
count : 2
```

type 2를 선택하면 연도를 입력받아 해당 연도에 가장 많은 package를 주문한 customer의 id와 이름 그리고 주문한 개수가 출력되게 하였습니다. 해당 쿼리 확인을 위해 출력한 인물 이외의 모든 customer는 package를 하나만 주문한 것으로 data를 만들었고 그 결과 올바르게 출력됨을 확인할 수 있었습니다.

```
char query2[300] = "with T(c, id) as (select count(customer_id), customer_id from pay where
year(paid_at)=";
strcat(query2, year);
strcat(query2, " group by customer_id) select id, c, C.c_name from T, customer C where c =
(select max(c) from T) and C.customer_id=id ";
```

해당 쿼리를 위해 작성한 코드는 위와 같습니다. 개수를 세기 위해 count와 group by 문법을 사용해 주었습니다. 또한 코드의 간결성을 위해 with as를 사용해 주었습니다.

## -Type 3

type 3를 선택하면 년도를 입력받아 해당 년도에 가장 많은 돈을 쓴 customer의 id와 이름 그리고 가격이 출력되게 하였습니다. 해당 쿼리 확인을 위해 가장 많은 돈을 쓴 인물과 가장 많은 택배를 주문한 인물을 다른 인물로 설정하였고 올바른 결과가 출력됨을 확인할 수 있었습니다.

```

Select tye : 3

---- TYPE III ----
** Find the customer who has spent the most money on shipping in the past year **

Which year? (yyyy format) : 2023

customer_id : C54248
customer name : sukgi
amount : 500000

```

```

char query3[300] = "with T(c, id) as (select sum(B.amount), P.customer_id from bill B, pay P
where P.bill_id=B.bill_id and year(paid_at)= ";
strcat(query3, year);
strcat(query3, " group by customer_id) select id, c, C.c_name from T, customer C where c =
(select max(c) from T) and id=C.customer_id");

```

해당 쿼리를 위해 작성한 코드는 위와 같습니다. 가격확인을 위해 sum과 group by 문법을 사용해 주었습니다. 또한 코드의 가독성을 위해 with as를 사용해 주었습니다.

#### -Type 4

```

Select tye : 4

---- TYPE IV ----
** Find the packages that were not delivered within the promised time **

package_id : P10005
package_id : P10006
package_id : P11135
package_id : P35201

```

type 4를 선택하면 예상시간보다 늦게 도착한 package의 id를 출력하게 됩니다. 이 쿼리를 확인하기 위해 예상시간보다 일찍 도착한 택배와 늦게 도착한 택배를 섞어 data를 생성했습니다. 올바른 결과가 출력됨을 확인할 수 있었습니다.

```

char query4[300] = "select P.package_id from tracking T, package P where T.package_id =
P.package_id and T.location=P.r_address and T.tracking_time>P.estimated_time";

```

해당 쿼리를 위해 작성한 코드는 위와 같습니다. 쿼리를 구현하기 위해 tracking의 location과 수령 주소가 같을 때 tracking\_time>estimated\_time인 경우를 select 했습니다.

## -Type 5

type 5를 선택하면 고객의 이름과 연도, 월을 입력하고 3가지 종류의 청구서를 출력하게 됩니다. 저는 subtype으로 세가지 영수증 중 하나를 선택할 수 있게 구현했습니다.

```
Select tye : 5

---- TYPE V ----
** Generate the bill for each customer for the past month **

Enter the customer name : heejin
Which year? (yyyy format) : 2023
Which month? (mm format) : 06

---- SUBTYPES IN TYPE I ----
1. TYPE V-1 : A simple bill: customer, address, and amount owed.
2. TYPE V-2 : A bill listing charges by type of service.
3. TYPE IV-3 : An itemize billing listing each individual shipment and the charges for it.

Select subtype : 1

customer name : heejin
customer id : C30094
address(billing) : mapo-gu
amount : 220000
```

1번 청구서는 인물에 대한 청구서입니다. 입력한 고객의 이름과 id 그리고 청구서가 보내지는 주소 그리고 금액이 출력됩니다. 한 고객이 아무리 많은 택배를 주문했더라도 한 쌍의 정보가 뜰 것입니다.

이와 비교하기 위해 3번 청구서부터 살펴보겠습니다.

```
Select subtype : 3

customer name : heejin
package id : P10005
paid at : 2023-06-08 15:00:00
payment method : credit_card
amount : 20000

customer name : heejin
package id : P10006
paid at : 2023-07-08 15:30:00
payment method : credit_card
amount : 200000
```

방금 전 살펴본 1번 청구서와 동일인물에 대해 검색한 결과입니다. 1번 청구서는 인물에 대한 청구서이기 때문에 한 쌍의 정보만 출력된 반면, 3번 청구서는 품목화한 청구서이기 때문에

package에 대해서 출력됩니다. 즉 검색한 고객이 시킨 package의 수만큼의 쌍의 정보가 출력됩니다. 출력된 정보는 고객의 이름, package id, 지불시기, 지불수단, 지불 가격입니다.

2번 청구서는 서비스별로 출력해야 합니다. 서비스는 package별로 정해지므로 package별로 service를 결정짓는 요소들과 가격을 출력해 주었습니다.

```
Select subtype : 2

customer name : heejin
package_id : P10005
package content : a_normal
(includes _a : oversea shipping, includes _h : hazard content)
package_type : big_box
package_weight : 5
timeline : overnight
amount : 20000

customer name : heejin
package_id : P10006
package content : monitor
(includes _a : oversea shipping, includes _h : hazard content)
package_type : big_box
package_weight : 3
timeline : overnight
amount : 200000
```

구현한 코드는 아래와 같습니다. type 5의 쿼리는 단순히 from에 사용된 table의 공통된 부분을 where에서 연결해주는 간단한 코드입니다.

```
char query51[300] = "select C.customer_id, B.b_address , sum(B.amount) from customer C, pay P,
bill B where C.customer_id=P.customer_id and P.bill_id=B.bill_id and year(B.b_date)=";
strcat(query51, year);
strcat(query51, " and month(B.b_date)=");
strcat(query51, month);
strcat(query51, " and C.c_name=");
strcat(query51, name); strcat(query51, "");
strcat(query51, "group by C.customer_id, B.b_address");

char query52[300]="select P.package_id, S.pkg_content, S.pkg_type, S.pkg_weight, S.timeline,
B.amount from service S, package P, customer C, bill B where P.svc_id=S.svc_id and
C.customer_id=P.customer_id and B.package_id=P.package_id and year(B.b_date)=";
strcat(query52, year);
strcat(query52, " and month(B.b_date)=");
strcat(query52, month);
strcat(query52, " and C.c_name=");
```

```
strcat(query52, name); strcat(query52, "");
```

```
char query53[300] = "select B.package_id, P.paid_at, P.method, B.amount from customer C, pay  
P, bill B where C.customer_id=P.customer_id and P.bill_id=B.bill_id and year(B.b_date)=";
```

```
strcat(query53, year);
```

```
strcat(query53, " and month(B.b_date)=";
```

```
strcat(query53, month);
```

```
strcat(query53, " and C.c_name=";
```

```
strcat(query53, name); strcat(query53, "");
```