

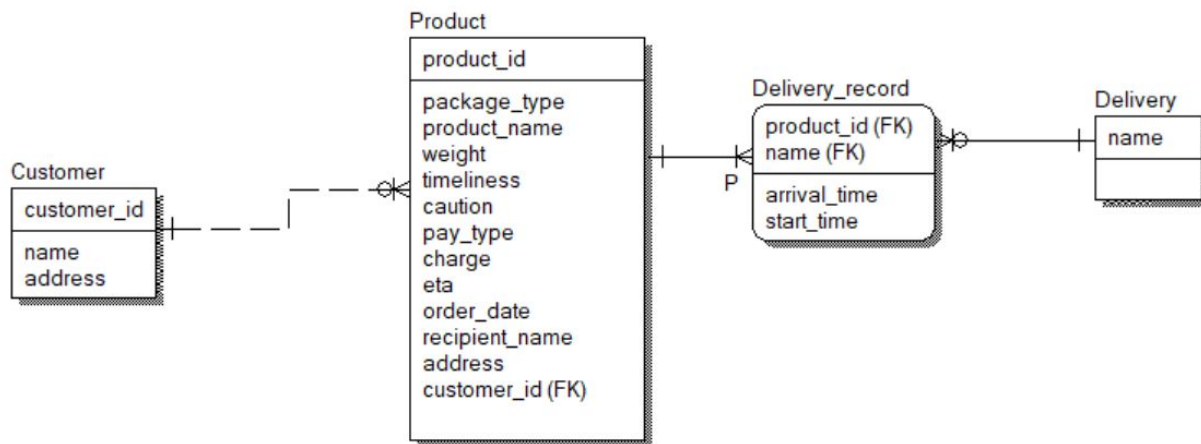
# Project 2 - Package Delivery System Report

- Normalization and Query Processing

20151550 박태준

## 1. BCNF Decomposition

### 1.1 Project 1에서 Relational Schema Diagram



위 그림은 이전 프로젝트에서 사용한 Relational Schema Diagram 입니다. 그러나, 해당 Schema가 BCNF 규칙을 어기지 않는다고 판단해 그대로 사용하기로 결정하였습니다.

#### Customer

Customer의 attribute에서 Functional Dependency를 정의할 경우,

(customer\_id, name, address)에서, customer\_id 를 알면 name, address를 모두 알 수 있기에, customer\_id -> name,address 의 관계를 갖습니다. 이는 Customer\_id 가 super key임을 만족하기에 BCNF를 위반하지 않습니다.

---

## Delivery

Delivery는 name attribute를 단일로 가집니다. 이 Table은 Delivery\_record의 운반 장소에 cardinality를 부여하기 위해 존재하며, 단일 attribute 이기 때문에 BCNF를 위반하지 않습니다.

## Delivery\_record

Delivery\_record는 (product\_id, name, start\_time, arrival\_time) 으로 구성됩니다. 각각의 delivery record의 한 row는 어떤 제품(product\_id) 이 어떤 배달 요소(name) 에 존재하고, 어느 시간부터 어느시간까지 존재했는지를 알려줍니다. 이때,

product\_id,name -> start\_time,arrival\_time 의 functional dependency만 존재합니다.

이는 product\_id,name이 super key이기 때문에 BCNF를 위반하지 않습니다.

## Product

Product는 (product\_id, package\_type, product\_name, weight, timeliness, caution, pay\_type, charge, eta, order\_date, recipient\_name, address, customer\_id) 로 구성되어 있습니다. 우선, primary key인 product\_id 를 알면 모든 attribute를 알 수 있기에, product\_id ->

package\_type, product\_name, weight, timeliness, caution, pay\_type, charge, eta, order\_date, recipient\_name, address, customer\_id

라는 functional dependency가 존재하며, 이는 BCNF를 만족합니다.

이후, functional dependency를 고민하는 과정에서, weight, package\_type, timeliness을 알 경우, charge를 알 수 있다는 생각을 하게 되었습니다. 이는, 우편물을 배송하기 위해 우체국을 방문하면, 요금을 책정하는 방식이 배달품의 종류(편지, 소포), 무게, 특급 배송 여부로 결정하는 것을 참고하였습니다.



왼쪽의 사진처럼, 소포를 보낼 때 무게를 측정하고, 또 편지인가 소포인가에 따라 금액이 바뀌며, 배달 기일을 짧게 정할수록 가격이 높아지는 경험을 참고한 것입니다. 이에 따라,

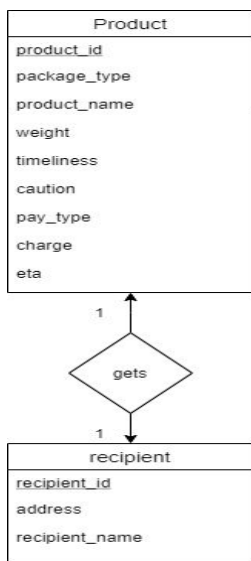
weight,package\_type,timeliness -> charge 라는 functional dependency를 찾았습니다.

이 attribute들은 superkey가 아니고 trivial한 관계가 아니기 때문에 BCNF를 만족하지 않았습니다. 따라서, Product를 Decompose 하였습니다.

Product(product\_id, package\_type, product\_name, weight, timeliness, caution, pay\_type, eta, order\_date, recipient\_name, address, customer\_id)

charge\_info(package\_type, weight, timeliness, charge)

따라서, 기존 Relational Schema Diagram에서 Product는 Product와 Charge\_info로 구성되었습니다.



※이외에도 만약 이전 프로젝트에서 ER Diagram에서 만들었던 recipient라는 table을 그대로 사용했다면, decompose가 필요했을 것입니다. 그러나, 1대1의 table을 합치는 과정에서 recipient에 대한 id를 삭제하였고, recipient\_id가 없어지면서 recipient의 정보 간의 functional dependency가 사라져 decompose를 할 필요가 없어졌습니다.

[Project 1의 E-R diagram의 일부]

## 1.2 변경한 Relational Schema Diagram

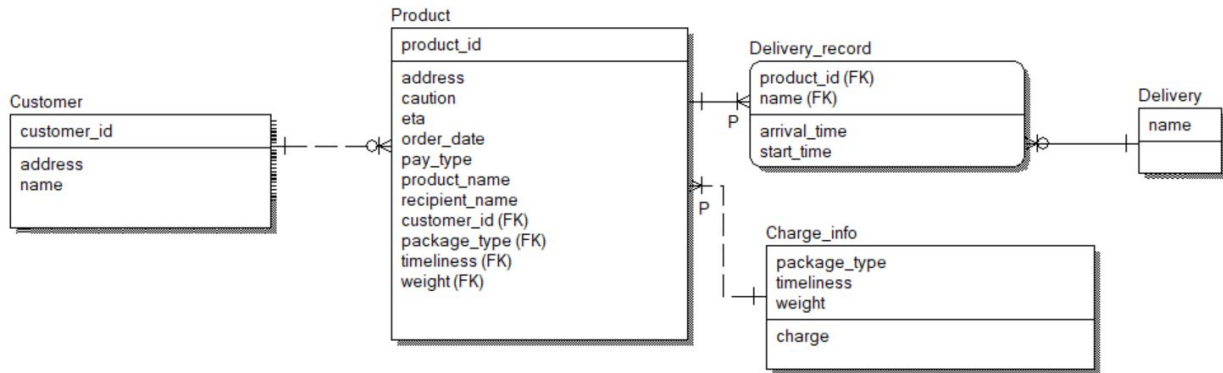
변경하지 않은 Table :

Customer, Delivery, Delivery\_record

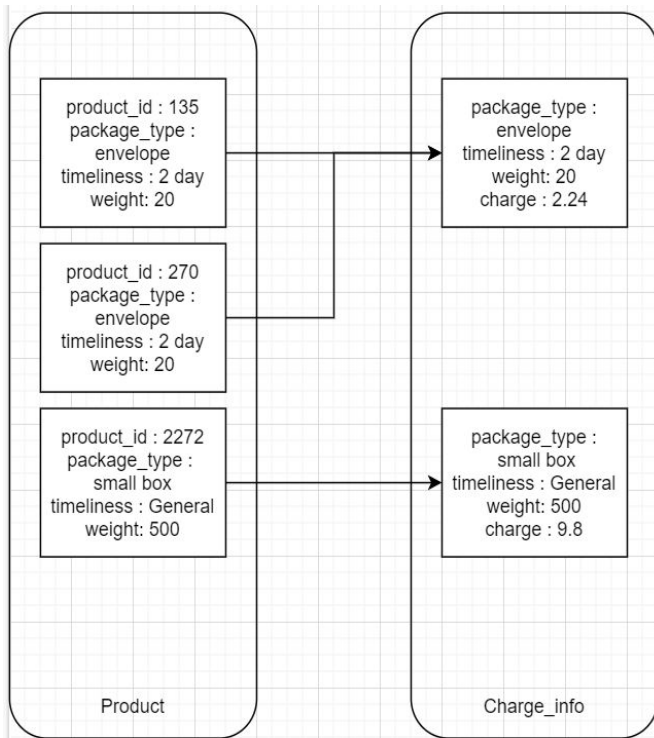
변경한 Table:

Product , Charge\_info (신규)

이를 바탕으로 새로 작성한 Relational Schema Diagram은 아래와 같습니다.

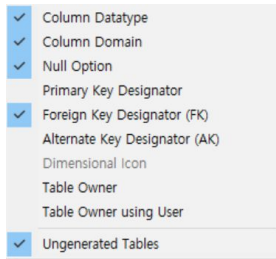


Product 와 Charge\_info 는 기존 같은 Table에서 decompose 되었습니다. 그런데, 소포가 어떤 것이냐 달라도 무게, 배송 기한, 패키지 타입이 같다면 요금은 같을 것이라고 상정하였습니다. 따라서 Product와 Charge\_info는 Many-to-One의 Cardinality를 가지게 됩니다. 이는 아래의 예시를 통해 쉽게 이해할 수 있습니다.



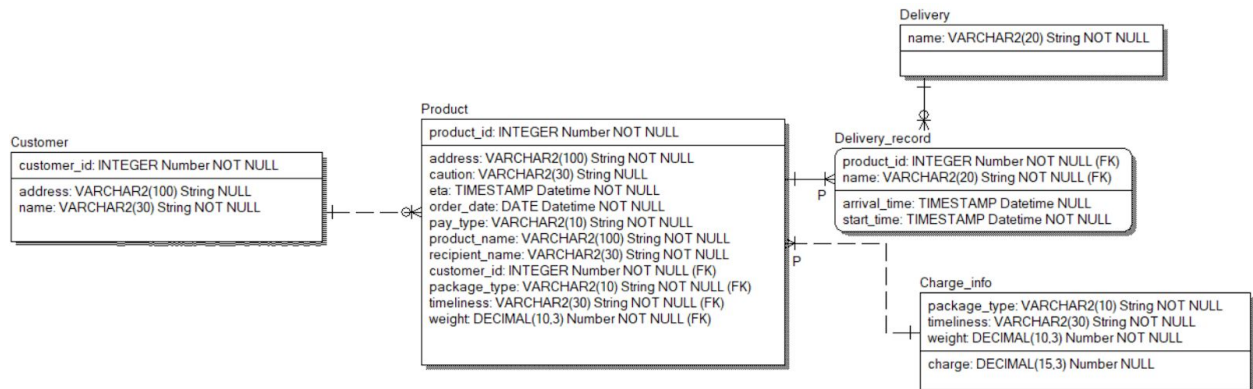
## 2. Physical Schema Diagram

BCNF 규칙을 따르도록 수정한 Schema Diagram의 Physical Schema Diagram을 구현하였습니다. 그러나, Physical Schema Diagram의 중요한 요소인 각 attribute의 속성값을 표시하기 위해 다음과 같은 display setting을 설정하였습니다.



이를 이용해 Physical Schema Diagram의 시각적 편의를 높였습니다.

구현한 Diagram은 아래와 같습니다.



### Customer

customer\_id : id는 모두 integer로 설정하였습니다. 또한 PK 들은 모두 Table의 값을 구분하게 해주는 값이니 자동으로 NOT NULL 처리 되었습니다.

address : 주소는 최대 100자의 스트링으로 처리하였으며, 이전 프로젝트에서 Customer 를 결제인으로 정의하였기 때문에, 결제인의 주소는 필요 없을 수 있다는 가정으로 NULL이 입력 가능하게 설정하였습니다.

name : 결제인의 이름은 최대 30자의 스트링으로 정하였으며, 이름은 필수이라고 생각해 NOT NULL로 설정하였습니다.

---

## Delivery

name : name은 배달 요소의 이름을 뜻합니다. (ex 'truck 1721', 'center 51', 'center suwon')  
이전 프로젝트에서 언급했듯, 이번 프로젝트에선 배달 요소의 종류와 번호를 분리하지 않아도 프로젝트를 진행하는데 어려움이 없기에 atomic한 요소로 설정하였습니다. 또한 pk값이기 때문에 NOT NULL 로 설정하였습니다.

## Delivery\_record

product\_id , name 은 Foreign Key 이기에 해당 attribute는 원 테이블에서 설명하였습니다.

start\_time : product\_id에 속하는 product가 해당 배달 요소에 도착한 시간을 의미합니다. 예를 들어, 물건이 3 집하장에 들어온다고 하면, (해당 id, 3 집하장) 의 start\_time 은 물건이 3 집하장에 들어온 시간을 의미합니다. 시간이기에 TIMESTAMP를 사용하였고, 무조건 물건이 그 곳에 경유되어야만 기록이 생성되기 때문에, NOT NULL로 설정하였습니다.

arrival\_time : 위의 start\_time과 상반되는 attribute로써, 해당 배달 요소에서 떠난 시간을 의미합니다. 예를 들어, 물건이 172 truck으로 운송되어 운송이 완료되어 다른 곳으로 옮겨졌다면 이 옮겨진 시간을 arrival\_time로 설정합니다. 아직 물건이 운송이 완료되지 않았을 수 있기 때문에 NULL도 가능하게 설정하였습니다.

## Charge\_info

package\_type : 배달되는 물건이 어떤 물건인지 (ex. envelope ... ) 알리는 package\_type은 최대 10글자의 string으로 설정하였습니다. 만일 이를 Integer로 설정할 경우, integer와 string을 매칭하는 또 다른 테이블이 필요하다는 생각에 string으로 정했습니다. 또한 Charge\_info의 primary key에 속하기에 NOT NULL로 설정되었습니다.

timeliness: 배달되는 물건이 언제까지 배달되길 원하는지 정하는 attribute인 timeliness는 최대 30자의 string으로 설정하였습니다. 이는 Overnight, second day 등 기본적인 옵션과, 1달, 2달 이내, 정확히 어느 일자까지, 라는 특수한 경우를 기입하기 위해 30자의 string으로 설정한 것입니다. 또한 NOT NULL이기 때문에, 특별한 주문이 없다면 general 이라는 string을 기입한다고 정의하였습니다.

---

weight : 무게는 소수점 3자리까지 저장하는 DECIMAL을 설정하였습니다. 무게는 이진법으로 저장하다가 증감이 발생하면 안되기에 DECIMAL 자료형을 사용하였습니다. 또한, 소수점 3자리까지 받은 상태에서 g 단위로 수 톤 단위까지 받기 위해 10자리로 설정하였습니다.

charge : 청구 요금또한 증감 문제가 발생하지 않기 위해 DECIMAL 자료형을 사용하였으며, 무게보다 현실적인 편차 범위가 넓기 때문에 좀 더 높은 15자리까지 받도록 설정하였습니다.

## Product

customer\_id, package\_type, timeliness, weight 는 타 테이블에서 가져온 Foreign Key이기에 다른 테이블에서 설명하였습니다.

product\_id : product를 구분하는 PK이자 각 배달품의 번호이기에 INTEGER로 설정하였습니다.

address : address는 배달품의 배송 주소로, 이전 결제인으로부터 받는 address와 동일한 크기의 자료형을 사용하였습니다. 또한 배송지가 없으면 안되기 때문에 NOT NULL로 설정하였습니다.

caution : 취급 주의사항은 30자의 string으로 구성하였는데, 취급 주의나 배달 요청사항 등은 종종 생략되는 경우도 있기에 NULL로 설정하였습니다.

eta : 예상 배송시간은 TIMESTAMP를 이용하였고, 현재 대부분의 서비스 업체에서 보통 주문시 예상 기간 알려주기 때문에 NOT NULL로 설정하였습니다.

**무료배송** (로켓프레시 상품 15,000원 이상 구매 시)

**내일(일) 새벽 7시 전 도착 보장** (밤 12시 전 주문 시) [ 예상 배송 시간 예시 ]

order\_date : 주문 날짜는 주문이 성사되는 즉시 생성되기 때문에 NOT NULL인 TIMESTAMP로 정의하였습니다.

pay\_type : 결제 방식도 배송 주문시 결정되기 때문에 NOT NULL을 가지는 10 사이즈의 스트링으로 설정하였습니다.

product\_name : 꼭 제품이 아니더라도 소포를 보낼 때 내용물이 무엇인지 기입하는 사항이 있었던 점을 참고하여 NOT NULL인 최대 100 길이의 string으로 설정하였습니다.

recipient\_name: 인터넷 주문시 수신자를 기입하지 않을 시 배달이 불가능했던 경험을 바탕으로 NOT NULL인 최대 30 길이로 결제자 정보의 name과 동일한 string으로 설정하였습니다.



---

## 3.Queries and Program

이번 프로젝트에선 ODBC를 이용해 직접 Query를 사용해 결과를 출력해보았습니다.

### 3.1 Table Creation, Tuple Insertion

ODBC를 이용해 Database를 연결하고, 이후 테이블을 생성, 튜플을 생성하였습니다. 이를 위해, sql문들을 모은 "20151550.txt" 파일을 생성하였습니다. 해당 파일은 모두 sql문으로 구성되어 있으며, 예시는 아래와 같습니다.

- ```
CREATE TABLE `delivery`.`customer` (  
  `customer_id` INT NOT NULL,  
  `address` VARCHAR(100) NULL,  
  `name` VARCHAR(30) NOT NULL,  
  PRIMARY KEY (`customer_id`));
```
- ```
CREATE TABLE `delivery`.`delivery` (  
  `name` VARCHAR(20) NOT NULL,  
  PRIMARY KEY (`name`),  
  UNIQUE INDEX `name_UNIQUE` (`name` ASC) VISIBLE);
```

Create 예시

```
use delivery;  
insert into delivery values ('recipient');  
insert into delivery values ('truck 1721');  
insert into delivery values ('truck 0733');  
insert into delivery values ('truck 2277');  
insert into delivery values ('ship 3084');
```

Insert 예시

이후, ODBC를 로컬 데이터베이스에 연결하고, 위의 sql 파일들을 읽어 실행시켜 테이블을 생성, 삽입하였습니다.



---

### 3.2 Program

ODBC 파일은 Main 함수만을 사용하도록 설계하였습니다. 프로그램의 사용 방식은 Mysql 데이터베이스 서버를 연결하고, 새 데이터베이스를 만드는 것으로 시작해 CRUD 쿼리들로 Table을 정의하고 Tuple들을 삽입해 DB를 만듭니다. 이후 미리 시스템에서 사용될 Query를 적어놓고, input을 받아 처리하고, 관련 tuple을 찾는 Query를 보내 connection에 사용하는 방법을 사용하였습니다.

먼저, database를 연결해 localhost에 root 유저가 sogangsp라는 비밀번호를 가지는 Mysql 서버를 연결하고, 'create database delivery' 라는 쿼리를 보내 delivery라는 데이터베이스를 만들었습니다.

Query를 connection에 사용하는 방법은,

1. MYSQL 객체를 할당하기 위한 초기화 과정으로서, mysql\_init() 함수를 호출합니다.
2. mysql\_real\_connect 를 이용해 DB와 커넥션을 만듭니다.
3. connect된 DB에 mysql\_query(connection, query) 를 호출해 DB에 원하는 SQL query를 보내 작동시킵니다.
4. Create, Update, Delete의 경우 결과를 가져올 필요가 없지만, Read의 경우 C에서 결과를 필요로 하는 경우가 있습니다. 따라서, select 문 등 query의 결과를 필요로 할 경우 mysql\_store\_result(connection) 함수를 이용해 저장합니다. 이후, mysql\_fetch\_row를 이용해 한줄씩 결과를 C에 가져올 수 있습니다.
5. 이때, select에 받은 순서대로 배열 형태로 가져옵니다.

위 과정을 이용해 필요한 정보 코드를 실행시킬 경우 화면은 다음과 같습니다.

```
*****
** Sogang Package Delivery Management System **
*****
----- SELECT QUERY TYPES -----

1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE V
0. QUIT
Any Input not in 0-5 will be ignored.
```

---

1번 타입을 실행시킬 때 화면입니다.

```
Which type of query? : 1
---- TYPE 1 ----
Input the number of truck : 0335
truck 0335 is not destroyed
---- TYPE 1 ----
Input the number of truck : 1721
----- Subtypes in TYPE 1 -----
    1. TYPE 1 - 1.
    2. TYPE 1 - 2.
    3. TYPE 1 - 3.
    0. QUIT.
Which type of query?
```

화면 1번에서는 트럭 중 사고가 난 트럭의 번호를 요구합니다.

만약 사고가 난 트럭이 아니라면, 사고가 나지 않았음을 알려줍니다. 1721 트럭은 사고가 발생했었는데, 1721의 번호를 입력하면, Subtype 화면으로 넘어갑니다. 이를 위해 사용한 쿼리는 다음과 같습니다.

```
select distinct(name) from delivery_record
```

```
where name != 'recipient'
```

```
and arrival_time is NULL;
```

저는 배송 기록을 만들 때, 배달이 완료되면 recipient 라는 name을 갖는 record를 만들게 했고, 이 레코드만이 arrival\_time이 NULL을 갖도록 했습니다. 그런데, 만약 사고가 발생한다면, arrival\_time이 NULL인데 recipient는 아닌 record가 생성될 것입니다. 이를 이용해 사고가 난 차량을 찾을 수 있습니다.

```

---- TYPE I ----
Input the number of truck : 1721
----- Subtypes in TYPE I -----
  1. TYPE I - 1.
  2. TYPE I - 2.
  3. TYPE I - 3.
  0. QUIT.
Which type of query? 1
---- TYPE I-1 ----

** Find all customers who had a package on the truck at the time of the crash.**
Customer name : customername3 id : 3

```

Subtype 1번의 결과입니다.

Subtype 1번은 사고가 났던 트럭에 물건이 있던 고객을 찾는 쿼리입니다.

이를 위해 사용한 쿼리는

```
select name,customer_id from customer
```

```
where customer_id =
```

```
(select distinct(p.customer_id)
```

```
from product as p inner join delivery_record as d on p.product_id=d.product_id
```

```
where d.name='truck 1721'
```

```
and d.arrival_time is NULL);
```

이때, truck 1721은 subtype 이전 쿼리에서 추출하여 sql문을 만들때 strcat 문을 이용해 합쳐서 사용하였습니다.

```

----- Subtypes in TYPE I -----
  1. TYPE I - 1.
  2. TYPE I - 2.
  3. TYPE I - 3.
  0. QUIT.
Which type of query? 2
---- TYPE I-2 ----

** Find all recipients who had a package on that truck at the time of the crash.**
Recipient : rec1
Recipient : rec2

```

---

subtype 2번의 결과입니다.

subtype 2번은 사고가 난 트럭에 배달품을 받을 고객의 리스트를 추출하는 쿼리입니다.

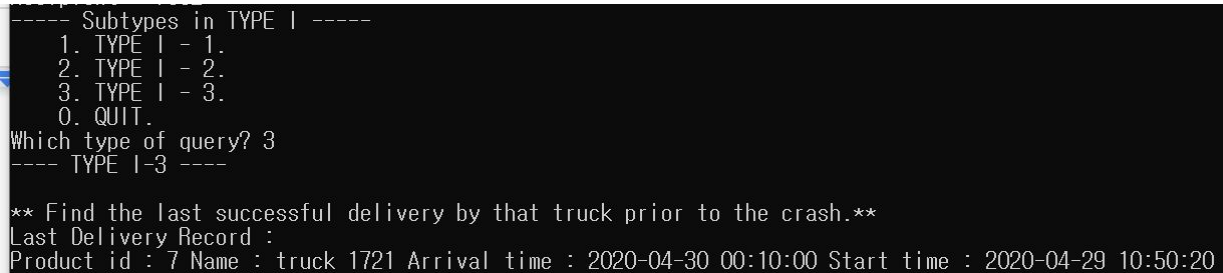
이를 위해 사용한 쿼리는

```
select distinct(p.recipient_name)
```

```
from product as p inner join delivery_record as d on p.product_id=d.product_id
```

```
where d.name= "truck 1721" and d.arrival_time is NULL;
```

입니다.



```
----- Subtypes in TYPE I -----
1. TYPE I - 1.
2. TYPE I - 2.
3. TYPE I - 3.
0. QUIT.
Which type of query? 3
----- TYPE I-3 -----

** Find the last successful delivery by that truck prior to the crash.**
Last Delivery Record :
Product id : 7 Name : truck 1721 Arrival time : 2020-04-30 00:10:00 Start time : 2020-04-29 10:50:20
```

subtype 3번의 결과입니다.

subtype 3번은 사고가 나기 전까지 가장 성공적으로 실행한 배달을 추출하는 쿼리입니다.

이때, 저는 delivery\_record 테이블로 배달 기록을 관리하며, 운송이 완료되면 arrival time이 존재하기에, 사고가 있기 전 배달기록 중 arrival time이 가장 최근인 기록을 찾으면 됩니다.

사용한 쿼리는 아래와 같습니다.

```
select * from delivery_record
```

```
where name= "truck 1721" and arrival_time <
```

```
(select distinct(start_time) from delivery_record where name="truck 1721"
```

```
and arrival_time is NULL)
```

```
order by start_time desc limit 1;"
```

```

----- SELECT QUERY TYPES -----
1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE V
0. QUIT
Any Input not in 0-5 will be ignored.
Which type of query? : 2
---- TYPE II ----

** Find the customer who has shipped the most packages in certain year**
Which Year? : 2019
Customer name : customername3 id : 3 total order : 1
---- TYPE II ----

```

Type 2의 실행결과 입니다.

특정 해에 대해서 가장 많이 배달을 주문한 고객을 찾는 쿼리입니다. 2019년에 대해선 customername3이라는 고객이 1번으로 가장 많은 주문을 했습니다.

```

** Find the customer who has shipped the most packages in certain year**
Which Year? : 2020
Customer name : customername2 id : 2 total order : 2
---- TYPE II ----

** Find the customer who has shipped the most packages in certain year**
Which Year? :

```

반면에, 2020년의 경우 customername2가 2번으로 가장 많은 주문을 했습니다.

이를 위한 쿼리는 아래와 같습니다.

```

select c.name,c.customer_id,count(p.customer_id)

from customer as c join product as p on c.customer_id = p.customer_id

where year(p.order_date)= 2019

group by p.customer_id order by count(p.customer_id) desc limit 1";

```

```

----- SELECT QUERY TYPES -----
1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE V
0. QUIT
Any Input not in 0-5 will be ignored.
Which type of query? : 3
---- TYPE III ----

** Find the customer who has spent the most money on shipping in the past certain year**
Which Year? : 2019
Customer Name : customername7 ID : 7

```

Type 3에 대한 실행결과 입니다.

type 3은 선택한 해에서 가장 많은 돈을 소비한 고객을 찾는 쿼리입니다.

2019년에는 customername7이라는 고객이 가장 많은 돈을 사용했습니다.

```

---- TYPE III ----

** Find the customer who has spent the most money on shipping in the past certain year**
Which Year? : 2020
Customer Name : customername8 ID : 8
---- TYPE III ----

** Find the customer who has spent the most money on shipping in the past certain year**
Which Year? :

```

2020년에는 customername8이 가장 많은 돈을 사용한 것을 확인할 수 있습니다.

이를 위해 사용한 쿼리는 아래와 같습니다.

**select name, customer\_id from customer**

**where customer\_id =**

**(select customer\_id**

**from product as p inner join charge\_info as c on p.package\_type = c.package\_type and  
p.timeliness = c.timeliness and p.weight = c.weight**

**where year(order\_date)= 2019**

---

```
group by customer_id order by sum(charge) desc limit 1)";
```

```
----- SELECT QUERY TYPES -----
1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE V
0. QUIT
Any Input not in 0-5 will be ignored.
Which type of query? : 4
---- TYPE IV ----

** Find those packages that were not delivered within the promised time**
Late Delivered Product's ID : 6 9
```

Type 4에 대한 실행결과입니다.

Type 4는 예정 시간보다 (estimated time arrival) 늦게 배송된 배달 건수를 출력하는 쿼리입니다.

실행 결과를 보시면, 배달의 id가 6, 9인 배달이 출력되는 것을 확인하실 수 있습니다.

이를 위한 쿼리는 다음과 같습니다.

```
select p.product_id
```

```
from product as p inner join delivery_record as d on p.product_id = d.product_id
```

```
where d.name = 'recipient' and d.start_time > p.eta";
```



```

----- SELECT QUERY TYPES -----
1. TYPE I
2. TYPE II
3. TYPE III
4. TYPE IV
5. TYPE V
0. QUIT
Any Input not in 0-5 will be ignored.
Which type of query? : 5
---- TYPE V ----

** Generate the bill for each customer for the past certain month.**
Customer Id : 15
Which month(YYYY-MM)? 2020-03
Generating Bill..
There was no matching information on database

```

마지막으로 Type 5에 대한 결과입니다.

Type 5는 Customer를 위해 영수증을 만드는 쿼리입니다.

예시에서는 Customer Name을 받아서 사용했는데, 저는 Customer Name이 중복될 수 있다고 가정했기 때문에, Customer 각각을 구분하기 위해서 Customer ID를 이용했습니다. Customer ID를 입력받고, 이 고객이 어느 해의 어느 월에 대한 영수증을 받고 싶은지 입력받으면, 이에 대한 영수증을 만듭니다. 그런데, 위의 그림처럼, 15번 고객이 2020-03에 거래 기록이 없을 경우, 거래 기록이 없다고 출력합니다.

```

** Generate the bill for each customer for the past certain month.**
Customer Id : 3
Which month(YYYY-MM)? 2020-05
Generating Bill..
Generating Done!
---- TYPE V ----

```

반면, 거래 기록이 있을 경우, 영수증을 만듭니다.

영수증을 모두 작성하면, Generating Done이라는 문구를 출력합니다. 그리고 bill.txt 라는 파일을 작성합니다.

bill.txt의 형태는 아래와 같습니다.

---

Customer ID	Name	Address	Amount
3	customername3	addressofcustomer3	\$ 9,5000

-----  
Itemized Billing List

=====

product_id	product_name	charge	pay_type	address	recipient_name	timeliness	package_type	weight	order_date	caution
3	pr3	7,000	creditcard	destinationofproduct3	rec1	a month	envelope	100.000	2020-05-21 09:50:20	need care
4	pr4	2,500	prepaid	destinationofproduct4	rec2	general	envelope	20.000	2020-05-10 13:50:20	fragile

=====

이처럼, 사용자의 개인 정보와 총액, 그리고 개별 기록에 대한 상세 사항이 출력됩니다.

이를 위해 사용한 쿼리는 아래와 같습니다.

-영수증의 상세 사항을 위한 쿼리

```
select product_id, product_name, charge, pay_type, address, recipient_name,  
timeliness, package_type, weight, order_date, caution
```

```
from product natural join charge_info
```

```
where customer_id = 3 and year(order_date)= 2020 and month(order_date)= 05
```

-영수증에 가입할 고객 기본 정보를 위한 쿼리

```
select customer_id,name,address from customer where customer_id = 3
```