

과목명: 데이터베이스시스템

<<Project #2>>

서강대학교 [컴퓨터공학과]

[20161566]

[권형준]

목 차

1. BCNF Decomposition

- 1.1 Simplified Test
- 1.2 Decomposing 'Package'
- 1.3 추가적인 수정사항
- 1.4 최종 Logical Schema

2. Physical Schema

- 2.1 Bills
- 2.2 Choose_Service
- 2.3 Customer
- 2.4 Recipient
- 2.5 Package
- 2.6 Move
- 2.7 Store
- 2.8 Warehouse
- 2.9 Transportation
- 2.10 Stops

3. Insert

- 3.1 Bills
- 3.2 Choose_Service
- 3.3 Customer
- 3.4 Recipient
- 3.5 Package
- 3.6 Move
- 3.7 Transportation

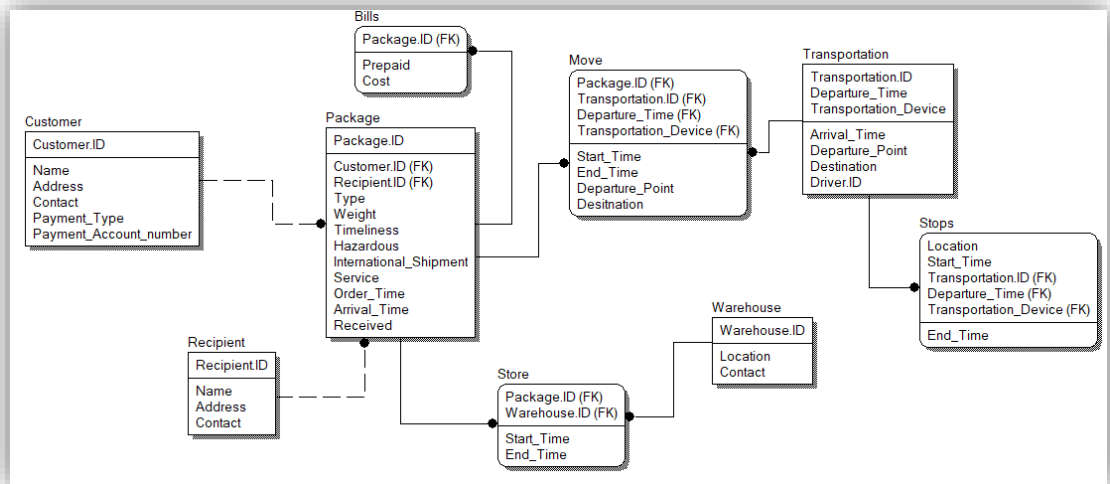
4. Query

- 4.1 Type-1.1
- 4.2 Type-1.2
- 4.3 Type-1.3
- 4.4 Type-2
- 4.5 Type-3
- 4.6 Type-4
- 4.7 Type-5

5. C-Code

- 5.1 Flow chart
- 5.2 '20161566.cpp'
- 5.3 '20161566_1.txt'
- 5.4 '20161566_2.txt'

1. BCNF Decomposition



기존의 Relational schema는 위와 같은 형태를 띄고 있었다. 7단원 수업시간에 배운 'BCNF Simplified Test'를 통해서 어떤 relation이 BCNF를 만족하지 않는지 확인하고 이를 BCNF로 decompose해보자. 'Simplified Test'는, 보통 BCNF를 만족하는지 확인하기 위해서는 F^+ (Closure)를 구해야 되지만, Decomposition이 이루어지지 않은 초기에는 F 만을 가지고 BCNF를 확인할 수 있는 test이다.

1.1 Simplified Test

- **Bills**
Bills relation에서는 (Package.ID \rightarrow Prepaid, Cost)로의 FD만이 존재한다. Package.ID가 Primary key임으로 BCNF를 만족한다.
- **Customer**
(Customer.ID \rightarrow Customer(relation))의 FD만이 존재하고, Customer.ID가 primary key임으로 BCNF를 만족한다.
- **Recipient**
(Recipient.ID \rightarrow Name, Address, Contact)의 FD만이 존재하고, Recipient.ID가 primary key임으로 BCNF를 만족한다.
- **Store**
(Package.ID, Warehouse.ID \rightarrow Start_time, End_time)의 FD만이 존재한다. 한번의 배송 과정에서 Package가 같은 장소에 2번이상 보관되는 일은 없다고 가정하였다. 이 FD 또한 좌변이 Primary key임으로 BCNF를 만족한다.
- **Warehouse**
(Warehouse.ID \rightarrow Location, Contact)의 FD만이 존재한다. Warehouse.ID가 primary key임으로 BCNF를 만족한다.
- **Stops**
이 relation 또한 primary key에서 출발하는 FD밖에 존재하지 않아 BCNF를 만족한다.
- **Transportation**
Primary key에서 출발하지 않는 FD가 있는지 확인해보자. ArrivalTime은 사용하는 운송수단과 그 운송수단의 출발시간을 알면 자동으로 정해진다. Departure_Point도 마찬가지로 운송수단과 출발시간을 알면 자동으로 정해진다. 같은 곳에서 다른 시간에 출발할 수 있어도 같은 시간에 다른 곳에서 출발할 수는 없다. Destination과 Driver.ID도 같은 논리로 FD를 가지지 않는다.

따라서 유일하게 있는 FD는 (Transportation.ID, Departure_Time, Transportation_Device → Arrival_Time, Departure_Point, Destination, Driver.ID) 밖에 없고 이는 BCNF를 만족한다.

- **Move**

기본적으로 (Package.ID, Transportation.ID, Departure_Time, Transportation_Device → Start_Time, End_Time, Departure_Point, Destination)의 FD를 가지고 있다. 이외에 FD는 존재하지 않는다. 여기서도 가정은 하나의 package를 운송할 때 같은 운송수단을 두 번 이상 사용하지 않는다는 것이었다. 따라서 move도 BCNF를 만족한다.

- **Package**

Package에서 Package.ID로부터 출발하는 FD를 제외하고 단 한 개의 FD가 더 존재한다. 그것은 바로 (Type, Weight, Timeliness → Service)이다. 이는 <프로젝트 2>에서 명시된 조건을 통해서 알 수 있다.

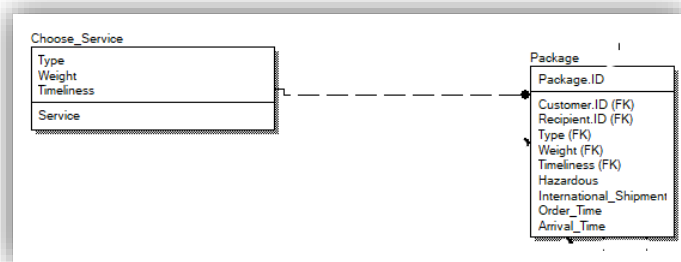
There are different kinds of service possibly based upon the type of package(flat envelope, small box, larger boxes), the weight of the package, and the timeliness of delivery (overnight, second day, or longer).

따라서 Package는 BCNF를 만족하지 않고 이를 Decompose하는 과정이 필요하다.

1.2 Decomposing 'Package'

교재에 나와있는 방법으로 Decompose를 진행하였다. BCNF를 만족하지 않는 FD는 (Type, Weight, Timeliness → Service)로 기존의 Package를 이를 기준으로 decompose하면

1. (Type, Weight, Timeliness, Service)
 2. (Package.ID, Customer.ID, Recipient.ID, Type, Weight, Timeliness, Hazardous, International_Shipment, Order_Time, Arrival_Time, Received)
- 가 된다. 이렇게 나뉜 relation을 one-to-many로 연결하여 다음과 같이 나타냈다.



두개 이상의 package가 같은 type, weight, timeliness를 가질 수는 있지만 하나의 package가 두개 이상의 service를 가지는 것은 불가능하여 one-to-many로 설정하였고, 적어도 하나의 service에 포함되어야 되기 때문에 not-null로 설정하였다.

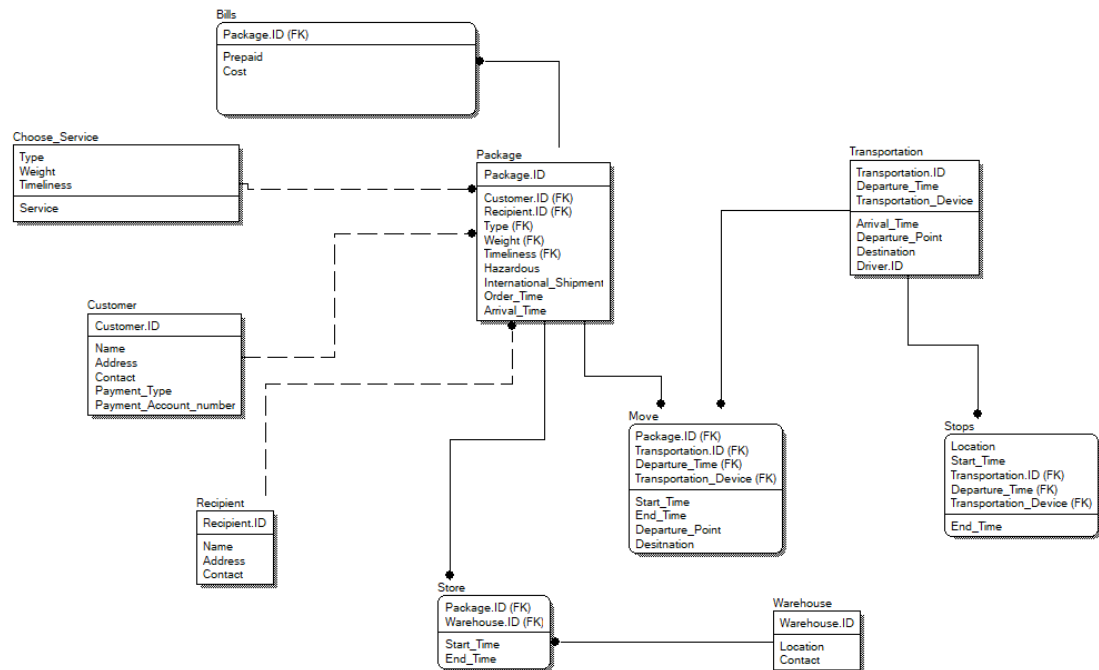
1.3 추가적인 수정사항

Decompose 말고도 이전의 logical schema에서 오류(?)를 발견하여 몇 가지 수정하였다.

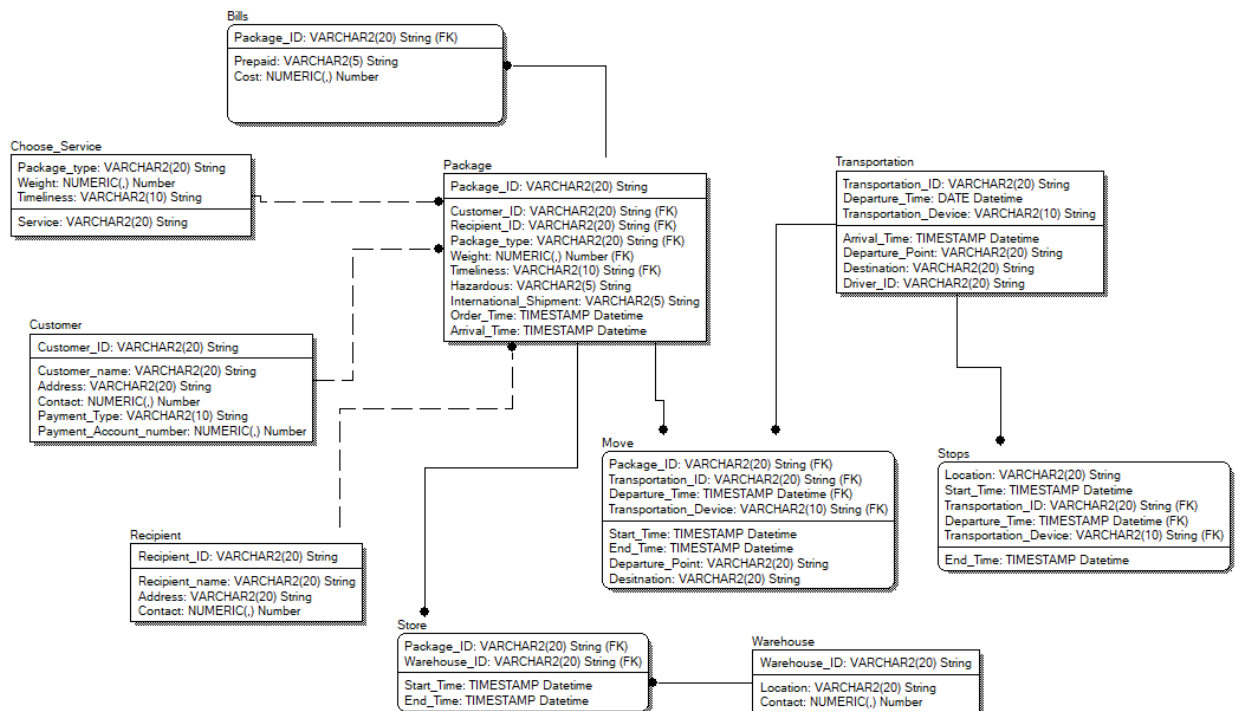
1. Attribute로 '~.ID'와 같은 형태로 표현하면 query에서 ambiguous하여 '.'을 '_'로 대체하였다.
2. 'Package'에서 Received가 Arrival_Time이 null이냐 아니냐로 판단할 수 있어서

data redundancy를 유발한다고 판단하여 삭제하였다.

1.4 최종 Logical Schema



2. Physical Schema



2.1 Bills

```
create table Bills(  
    Package_ID varchar(20),  
    Prepaid varchar(5) check(Prepaid='Yes' OR Prepaid='No'),  
    Cost numeric(15,5),  
    primary key (Package_ID),  
    foreign key (Package_ID) references Package(Package_ID)  
);
```

‘Bills’ relation은 대응하는 ‘Package’의 가격과 선불유무의 정보를 담고 있다. Prepaid로는 ‘Yes’또는 ‘No’ 두개의 값만이 들어올 수 있고, Cost에는 대응하는 가격을 총 15자리, 소수점 아래까지는 5자리까지 허용하였다. 이를 SQL로 표현하면 위와 같다.

2.2 Choose_Service

```
create table Choose_Service(  
    Package_type varchar(20) check (Package_type='flat envelope' OR Package_type='small box' OR Package_type='larger box'),  
    Weight numeric(10,3) check (Weight>0),  
    Timeliness varchar(10) check (Timeliness='overnight' OR Timeliness='second day' OR Timeliness='longer'),  
    Service varchar(20),  
    primary key (Package_type,Weight, Timeliness));
```

‘Choose_Service’는 주어진 type, weight, timeliness등에 따라서 해당하는 Service를 정해주는 relation이다. 명세서에 나와있는 조건을 바탕으로 Package_type으로는 ‘flat envelope’, ‘small box’, ‘larger box’만이 들어올 수 있고, Weight는 당연히 양수일 것이다. Timeliness도 명세서의 조건을 바탕으로 ‘overnight’, ‘second day’, ‘longer’만이 들어올 수 있게 설정하였다.

2.3 Customer

```
create table Customer(  
    Customer_ID varchar(20),  
    Customer_name varchar(20),  
    Address varchar(20),  
    Contact numeric(12,0),  
    Payment_Type varchar(10) check(Payment_Type = 'Infrequent' OR Payment_Type = 'Frequent'),  
    Payment_Account_Number numeric(20,0),  
    primary key(Customer_ID)  
);
```

Customer들을 구분하는 primary key로 Customer_ID라는 고유 식별 번호를 부여하였다. 또한 이름과 주소 그리고 연락처를 저장할 attribute을 설정하였고 연락처를 제외하고는 string으로 설정하였다. 주소에는 숫자 뿐만 아니라 문자도 들어가기 때문이다. ‘Payment Type’은 명세서에 주어진 조건에 맞춰서 ‘Infrequent’와 ‘Frequent’로 나누었다. 마지막으로 ‘Frequent’하게 지불을 하는 고객에 한해서는 ‘Payment Account Number’를 저장하게 하였고, 그 외의 고객, 즉 Infrequent한 고객은 Null값을 저장하게 하였다.

2.4 Recipient

```
create table Recipient(  
    Recipient_ID    varchar(20),  
    Recipient_name  varchar(20),  
    Address          varchar(20),  
    Contact          numeric(12,0),  
    primary key(Recipient_ID)  
);
```

‘Customer’와 저장된 정보가 거의 비슷하다. Recipient는 수령인의 정보를 저장한 것으로 고유 식별번호로 Recipient_ID를 부여하였다. 그 외의 attribute은 customer와 동일한 조건을 가진다. 이름으로 20개의 문자를 받을 수 있고, 주소도 마찬가지로 연락처로는 총 12자리의 소수를 포함하지 않은 수를 입력 받을 수 있다.

2.5 Package

```
create table Package(  
    Package_ID    varchar(20),  
    Customer_ID   varchar(20),  
    Recipient_ID  varchar(20),  
    Package_type  varchar(20),  
    Weight        numeric(10,3),  
    Timeliness    varchar(10),  
    Hazardous     varchar(5) check(Hazardous='Yes' OR Hazardous='No'),  
    International varchar(5) check(International='Yes' OR International='No'),  
    Order_Time    Timestamp,  
    Arrival_Time  Timestamp,  
    primary key(Package_ID),  
    foreign key(Recipient_ID) references Recipient(Recipient_ID)  
        on delete set null,  
    foreign key(Package_type, Weight, Timeliness) references Choose_Service(Package_type, Weight, Timeliness)  
        on delete set null,  
    foreign key(Customer_ID) references Customer(Customer_ID)  
        on delete set null  
);
```

Foreign key로 가져오는 attribute들을 제외하고 살펴보면, Hazardous와 International은 명세서의 조건을 표현하기 위해 추가하였다. 둘 다 ‘Yes’ 또는 ‘No’의 값 만을 가질 수 있다. Order_Time과 Arrival_Time은 Package를 주문하고 수령한 시간이며 Timestamp라는 자료형을 사용하였다. ‘Timestamp’자료형은 ‘0000-00-00 00:00:00’형태로 년도, 월, 일, 시간, 분, 초까지 표현이 가능한 자료형이다. 같은 자료형끼리 비교도 할 수 있어서 이후에 query를 수행할 때 편할 것 같아서 사용하였다. 초기에 도착하지 않은 Package의 Arrival_Time은 Null로 설정이 된다. 밑에 foreign key를 선언한 곳에서 ‘on delete set null’이라는 조건은 만약 부모 relation에서 해당하는 attribute이 삭제되었을 경우 자식 relation에서 삭제하지 않고 null로 설정한다는 뜻이다. 우리가 사용하는 query에서는 큰 의미가 없는 부분이지만 추가하였다. 예를 들어서 이후에 customer가 회원탈퇴를 하여 Customer_ID와 같은 정보들이 삭제되어도 과거에 배송 기록은 남아야 하기 때문에, 발송인을 null로 설정한다고 이해하면 될 것이다.

2.6 Move

```
create table Move(  
    Package_ID varchar(20),  
    Transportation_ID varchar(20),  
    Departure_Time timestamp,  
    Transportation_Device varchar(10),  
    Start_Time timestamp,  
    End_Time timestamp,  
    Departure_Point varchar(20),  
    Destination varchar(20),  
    primary key(Package_ID,Transportation_ID,Departure_Time, Transportation_Device),  
    foreign key(Package_ID) references Package(Package_ID),  
    foreign key(Transportation_ID,Departure_Time, Transportation_Device) references Transportation(Transportation_ID,Departure_Time, Transportation_Device)  
);
```

‘Move’라는 relation은 Package의 이동 정보를 담고 있다. 언제부터 언제까지 어떤 운송수단을 이용하여 어디까지 갔는지를 기록한다. Foreign key를 제외하고 살펴보면 Start_Time, End_Time은 해당하는 Package가 해당하는 운송수단을 언제부터 언제까지 사용했는지를 기록한다. 오후 1시부터 오후 5시까지 이동하는 Truck을 오후 2시부터 3시까지만 사용하여 이동하는 Package가 존재할 수 있기 때문에 필요한 기록이다. Start_Time과 End_Time의 data type은 timestamp를 사용하였다. Departure_Point와 Destination은 출발점과 도착점을 기록하면 20개의 문자를 받을 수 있게 설정하였다.

2.7 Store

```
create table Store(  
    Package_ID varchar(20),  
    Warehouse_ID varchar(20),  
    Start_Time timestamp,  
    End_Time timestamp,  
    primary key(Package_ID, Warehouse_ID),  
    foreign key(Package_ID) references Package(Package_ID),  
    foreign key(Warehouse_ID) references Warehouse(Warehouse_ID)  
);
```

‘Store’은 해당하는 Package가 해당하는 Warehouse에 언제부터 언제까지 보관되는지를 기록한다. ‘Move’와 마찬가지로 Start_Time, End_Time은 timestamp의 자료형으로 표현하였다.

2.8 Warehouse

```
create table Warehouse(  
    Warehouse_ID varchar(20),  
    Location varchar(20),  
    Contact numeric(12,0),  
    primary key(Warehouse_ID)  
);
```

여태까지 모든 ID들과 같이 Warehouse_ID도 varchar(20)으로 통일하였다. Warehouse의 위치를 저장하는 Location과 연락처를 저장하는 Contact가 있다.

2.9 Transportation

```
create table Transportation(  
    Transportation_ID varchar(20),  
    Departure_Time timestamp,  
    Transportation_Device varchar(10),  
    Arrival_Time timestamp,  
    Departure_Point varchar(20),  
    Destination varchar(20),  
    Driver_ID varchar(20),  
    primary key(Transportation_ID, Departure_Time, Transportation_Device)  
);
```

Transportation의 Primary key는 ID만으로는 부족하다. Truck과 Plane이 같은 ID를 가질 수 있다는 전제하에 이를 구분하기 위해 Transportation_Device도 필요하며 가장 중요한 Departure_Time이 필요하다. 같은 운송수단이라도 출발하는 시간에 따라서 완전히 다른 운송수단으로 봐야 되기 때문이다. ID와 장소를 나타내는 Departure_Point, Destination등은 varchar(20)을 사용하였고 날짜, 시간을 나타내는 Departure_Time, Arrival_Time은 timestamp 자료형을 사용하였다.

2.10 Stops

```
create table Stops(  
    Location varchar(20),  
    Start_Time timestamp,  
    Transportation_ID varchar(20),  
    Departure_Time timestamp,  
    Transportation_Device varchar(10),  
    End_Time timestamp,  
    primary key(Location, Start_Time, Transportation_ID, Departure_Time, Transportation_Device),  
    foreign key(Transportation_ID, Departure_Time, Transportation_Device)  
        references Transportation(Transportation_ID, Departure_Time, Transportation_Device)  
);
```

Stops는 명세서에는 나와있지 않지만 임의로 추가한 relationship이다. 운송수단이 중간중간 Package를 전달하거나 다른 package들을 추가하기 위해서 정차하는 시간이 있을 것이다. 이를 표현하기 위해서 'Stops'라는 relation을 추가하였다. 위에서 했던 것과 같이 시간을 나타내는 attribute들은 timestamp를 장소를 나타내는 attribute들은 varchar(20)을 사용하였다.

3. Insert

위에서 'create table'로 Physical Schema를 어떻게 구현하는지 알아보았다. 이제 작성한 table안에 data를 넣는 'insert into'과정이 필요하다. 작성한 database에는 table이 총 10개가 있지만, 이번 프로젝트에서 구현해야 될 총 7개의 query를 답하기 위해서는 7개의 table만 있어도 무관하다. 따라서 query에 참여하지 않는 'Stops', 'Warehouse', 'Store'에는 insert를 하지 않고 query에 참여를 하는 나머지 7개의 table에 insert하는 과정을 알아보자. Insert into는 형식도 정해져 있어서 설명할 것이 많지가 않다. 따라서 query에서 나타나는 data들을 중심으로 insert과정을 설명하겠다. 또한 '.txt'파일로 읽어서 실행하는 CRUD파일을 workbench에서 실행한 형태로 옮겨와서 가독성을 높였다.

3.1 Customer

```
insert into Customer values('20161566','Kwon Hyung Joon','sogang K-301','01023454321','Infrequent',NULL);
insert into Customer values('20161567','Chun He Jin','sogang R-912','01022020891','Infrequent',NULL);
insert into Customer values('20161568','Jung Jin One','sogang R-914','01044903304','Infrequent',NULL);
insert into Customer values('20161569','Kang Min Suck','sogang RA-993','01030294039','Infrequent',NULL);
insert into Customer values('20161570','Son','sogang H-101','01012340938','Infrequent',NULL);
insert into Customer values('20161571','Kim Su Hyun','sogang R-123','01030491009','Infrequent',NULL);
insert into Customer values('20161572','Jo Eun Gee','sogang D-303','01010930023','Infrequent',NULL);
insert into Customer values('20161573','Park Seung Joo','sogang D-301','01057684955','Infrequent',NULL);
insert into Customer values('20161574','Seul Gee','sogang K-401','01099807783','frequent','1034004924');
insert into Customer values('20161575','Ted Mosby','sogang AS-202','01029384820','frequent','20341435234');
insert into Customer values('20161576','Sheldon cooper','sogang R-505','01009877899','frequent','120349120348');
insert into Customer values('20161577','Rajesh Coothropaly','sogang J-101','01078976786','frequent','09839573023');
insert into Customer values('20161578','Kim so Jin','sogang J-303','01039483947','frequent','34219395982');
insert into Customer values('20161579','Anna Kendrick','sogang H-302','01039208937','frequent','0093927492701');
insert into Customer values('20161580','LeBron James','sogang R-912','01000098889','frequent','14148582393');
```

위는 총 15명의 Customer정보를 insert하는 과정이다. Payment type이 'Infrequent'인 사람들은 계좌번호가 들어가야 될 곳에 NULL이 들어간 것을 확인할 수 있다. Primary key인 Customer_ID만 중복되지 않으면 오류가 발생하지 않는다.

3.2 Recipient

```
insert into Recipient values('20161581','Sherlock Holmes','221B Baker Street'),'0432501');
insert into Recipient values('20161582','John Watson','221C Baker Street'),'125342341');
insert into Recipient values('20161583','Bilbo Bagins','221D Baker Street'),'32643543');
insert into Recipient values('20161584','Anthony Davids','221E Baker Street'),'365474374');
insert into Recipient values('20161585','Michael Jordan','221F Baker Street'),'74572855');
insert into Recipient values('20161586','Stephen Curry','221G Baker Street'),'564722818');
insert into Recipient values('20161587','Kevin Snake Durant','221H Baker Street'),'58679796');
insert into Recipient values('20161588','Kyre Irving','221I Baker Street'),'67836838');
insert into Recipient values('20161589','Klay Thompson','221J Baker Street'),'5856372');
insert into Recipient values('20161590','Kevin Love','221K Baker Street'),'231412341');
insert into Recipient values('20161591','Kawahi Leonard','221L Baker Street'),'3234623');
insert into Recipient values('20161592','Demar Derozan','221M Baker Street'),'2364248');
insert into Recipient values('20161593','Deric Rose','221N Baker Street'),'43685794');
insert into Recipient values('20161594','Super Nova','221O Baker Street'),'34756436');
insert into Recipient values('20161595','Ben Simmons','221P Baker Street'),'347457347');
```

Customer와 마찬가지로 총 15명의 Recipient 정보를 입력한다. Primary key인 Recipient_ID만 중복되지 않으면 된다.

3.3 Package

```
insert into Package values('19-0000001','20161580','20161586','larger box','84.3','overnight','No','No','2019-06-04 10:00:33',NULL);
insert into Package values('19-0000002','20161575','20161595','flat envelope','37.5','overnight','No','No','2019-06-04 10:00:33',NULL);
insert into Package values('19-0000003','20161576','20161583','small box','40','overnight','No','No','2019-06-04 14:34:33','2019-06-05 11:30:00');
insert into Package values('19-0000004','20161576','20161584','larger box','40','second day','Yes','No','2019-12-29 14:37:33','2019-12-31 11:30:00');
insert into Package values('17-0000001','20161572','20161585','small box','23','second day','No','Yes','2017-03-07 14:00:00','2017-03-10 14:00:00');
insert into Package values('18-0000001','20161570','20161590','small box','84.3','overnight','Yes','Yes','2018-02-09 12:00:00','2018-02-11 12:03:33');
insert into Package values('18-0000002','20161579','20161588','larger box','84.3','overnight','No','Yes','2018-02-13 16:30:00','2018-02-14 09:03:33');
insert into Package values('18-0000003','20161579','20161593','flat envelope','84.3','longer','Yes','No','2018-02-17 18:00:00','2018-02-19 12:50:33');
insert into Package values('18-0000004','20161579','20161592','flat envelope','37.5','overnight','Yes','Yes','2018-02-20 09:35:00','2018-02-21 15:03:33');
insert into Package values('18-0000005','20161579','20161582','larger box','40','second day','Yes','No','2018-02-27 12:00:00','2018-02-28 09:00:00');
insert into Package values('19-0000005','20161574','20161581','larger box','40','second day','No','No','2019-06-04 10:00:33',NULL);
insert into Package values('19-0000006','20161578','20161595','small box','55','second day','No','No','2019-06-04 10:00:33',NULL);
insert into Package values('17-0000002','20161572','20161584','small box','84.3','overnight','Yes','No','2017-03-13 10:30:00','2017-03-14 09:30:24');
insert into Package values('17-0000003','20161580','20161590','larger box','40','second day','Yes','No','2017-05-03 10:30:00','2017-05-06 12:30:00');
insert into Package values('17-0000004','20161580','20161593','larger box','24','overnight','No','No','2017-06-05 10:45:00','2017-06-07 12:00:00');
```

Package의 data를 insert하는 부분이다. Arrival_Time에 NULL값이 들어가 있는 Package들은 아직 도착을 하지 않은 Package들이다. 뒤에서 살펴볼 '1721 Truck'이 운송하고 있던 소포들이 여기에 해당한다. 또한 자세히 보면 Timeliness에 따라서 도착해야 될 시간에 도착하지 않은 package들도 위의 입력에서 골라낼 수 있다. 이는 이후 query부분에서 살펴보겠다.

3.4 Bills

```
insert into Bills values ('19-0000001','No','35000'); #lebron -> curry (crash)
insert into Bills values ('19-0000002','Yes','43000'); # ted -> ben (crash)
insert into Bills values ('19-0000003','No','20000'); #cooper->bilbo (last delivery)
insert into Bills values ('19-0000004','No','1500'); #cooper -> davids
insert into Bills values ('17-0000001','No','23000');#JoEunGee-->Michael Jordan (late)
insert into Bills values ('18-0000001','Yes','33445');#Son --> kevin love (late)
insert into Bills values ('18-0000002','Yes','340000');#anna kendrick -> kyre irving (for bill)
insert into Bills values ('18-0000003','Yes','230000'); # anna kendrick -> deric rose (for bill)
insert into Bills values ('18-0000004','Yes','1000'); #anna kendrick -> demar derozan (for bill)
insert into Bills values ('18-0000005','Yes','19000'); # anna kendrick -> john watson (for bill)
insert into Bills values ('19-0000005','Yes','42000'); #seul gee --> sherlock holms (crash)
insert into Bills values ('19-0000006','No','49000'); #kim so jin --> ben simmons (crash)
insert into Bills values ('17-0000002','Yes','21000'); #joe eun gee --> anthony davids (late)
insert into Bills values ('17-0000003','No','41000'); #lebron james --> kevin love (late)
insert into Bills values ('17-0000004','Yes','1000'); #lebron james --> deric rose (late)
```

Bill의 insert 과정이다. 옆에 주석으로 표시한 것은 각 query에 해당하는 data를 표현한 것이다. Bill은 Package와 one-to-one relationship을 가진다. Package뒤에 표현하기에는 너무 복잡해 보여서 Bill 뒤에 주석으로 표현하였다. (Crash)로 표현한 것은 Query-1에서 사용되는 사고가 발생하였을 때 truck에 타고 있던 package들을 나타낸 것이다. (last delivery)는 사고가 발생하기 전 성공적으로 전달한 마지막 Package를 나타낸다. (late)가 붙은 package들은 정해진 timeliness에 도착하지 못한 package들을 나타내고, (for bill)은 query 5에서 bill을 만들기 위해 특별히 표시한 것이다. 사실 query3, 4, 5에는 다른 입력이 와도 상관없지만, 가장 그 결과를 잘 나타내는 입력은 위에서 표현한 것들이다.

3.5 Transportation

```
insert into Transportation values('1721','2020-01-03 14:02:21','Truck','2020-01-03 17:00:03','SEOUL','BUSAN','20031');
insert into Transportation values('1721','2019-06-05 09:00:00','Truck',NULL,'SEOUL','BUSAN','20031');
insert into Transportation values('3443','1997-01-21 14:00:22','Plane','1997-01-22 15:00:03','CALIFORNIA','LONDON','412443');
insert into Transportation values('3443','2019-12-30 10:00:00','Train','2019-12-31 12:00:00','SEOUL','NAPHAEE','535124');
insert into Transportation values('4243','2017-03-06 12:33:44','Ship','2017-03-08 13:00:00','CALIFORNIA','HAWAII','341243');
insert into Transportation values('4453','2017-02-08 12:30:00','Truck','2017-02-12 14:00:03','MIAMI','MEXICOCITY','34129');
insert into Transportation values('0003','2018-02-13 18:30:00','Truck','2018-02-14 18:30:00','BUSAN','SEOUL','6993');
insert into Transportation values('0324','2018-02-17 20:00:00','Truck','2018-02-19 09:03:33','GANGRUNG','GYUNGJOO','7546');
insert into Transportation values('0988','2018-02-20 09:50:00','Truck','2018-02-21 12:30:00','SEOUL','BUSAN','9490');
insert into Transportation values('0288','2018-02-27 15:00:00','Truck','2018-02-28 09:03:33','PAJU','INCHEON','7748');
insert into Transportation values('2341','2017-03-12 18:00:00','Train','2017-03-15 09:30:24','MOSCOW','LONDON','33354');
insert into Transportation values('5432','2017-05-03 12:30:00','Ship','2017-05-06 12:30:00','JEJU','INCHEON','3414');
insert into Transportation values('3414','2017-06-06 10:45:00','Truck','2017-06-07 12:00:00','NEW YORK','TEXAS','97869');
insert into Transportation values('3665','2015-03-04 19:30:00','Truck','2015-03-05 13:00:00','BUSAN','NAMEHAE','12433');
insert into Transportation values('1992','2017-03-12 14:20:22','Ship','2017-03-15 19:00:00','HAWAII','TOKYO','1532');
```

‘Transportation’의 insert과정을 나타낸 것이다. 앞의 3개의 attribute이 모여서 primary key역할을 한다. 2번째 insert into를 살펴보면 도착시간이 NULL인 것을 확인할 수 있다. 이는 ‘Truck 1721’이 도중에 사고로 인해 도착하지 않아서 도착시간이 비어 있는 것이다.

3.6 Move

```
insert into Move values('19-0000001','1721','2019-06-05 09:00:00','Truck','2019-06-05 09:00:00', NULL,'SEOUL','BUSAN');
insert into Move values('19-0000002','1721','2019-06-05 09:00:00','Truck','2019-06-05 09:00:00', NULL,'SEOUL','BUSAN');
insert into Move values('19-0000003','1721','2019-06-05 09:00:00','Truck','2019-06-05 10:00:00', '2019-06-05 11:30:00','SEOUL','DAEGU');
insert into Move values('19-0000004','3443','2019-12-30 10:00:00','Train','2019-12-30 10:00:00', '2019-12-31 11:30:00','SEOUL','CHEUGJU');
insert into Move values('17-0000001','4243','2017-03-06 12:33:44','Ship','2017-03-07 15:00:00','2017-03-10 13:00:00','LA','HAWAII');
insert into Move values('18-0000001','4453','2017-02-08 12:30:00','Truck','2017-02-09 14:00:03','2018-02-11 09:03:33','MIAMI','WASHINGTON');
insert into Move values('18-0000002','0003','2018-02-13 18:30:00','Truck','2018-02-14 18:30:00','2018-02-14 09:03:33','BUSAN','SEOUL');
insert into Move values('18-0000003','0324','2018-02-17 20:00:00','Truck','2018-02-19 09:03:33','2018-02-19 12:50:33','GANGRUNG','GYUNGJOO');
insert into Move values('18-0000004','0988','2018-02-20 09:50:00','Truck','2018-02-21 12:30:00','2018-02-21 15:03:33','SEOUL','BUSAN');
insert into Move values('18-0000005','0288','2018-02-27 15:00:00','Truck','2018-02-28 09:03:33','2018-02-28 09:00:00','PAJU','INCHEON');
insert into Move values('19-0000005','1721','2019-06-05 09:00:00','Truck','2019-06-05 09:00:00', NULL,'SEOUL','BUSAN');
insert into Move values('19-0000006','1721','2019-06-05 09:00:00','Truck','2019-06-05 09:00:00', NULL,'SEOUL','BUSAN');
insert into Move values('17-0000002','2341','2017-03-12 18:00:00','Train','2017-03-12 18:00:00','2017-03-15 09:30:24','MOSCOW','LONDON');
insert into Move values('17-0000003','5432','2017-05-03 12:30:00','Ship','2017-05-03 12:30:00','2017-05-06 12:30:00','JEJU','INCHEON');
insert into Move values('17-0000004','3414','2017-06-06 10:45:00','Truck','2017-06-06 10:45:00','2017-06-07 12:00:00','NEW YORK','TEXAS');
```

Move는 특정 package와 특정 transportation의 관계를 나타낸다. Arrival_Time이 있어야 될 곳에 NULL이 있는 4개의 insert가 있다 이는 truck 1721에 탑승하고 있었던 것을 확인할 수 있고, truck 1721에 탑승하고 있었으면서 도착시간이 null이 아닌 것은 사고가 발생하기 전에 성공적으로 전달한 package라고 유추할 수 있다.

3.7 Choose_Service

```
insert into Choose_Service values('flat envelope', '37.5', 'overnight','service_1');
insert into Choose_Service values('flat envelope', '90', 'overnight','service_1');
insert into Choose_Service values('flat envelope', '39', 'second day','service_1');
insert into Choose_Service values('flat envelope', '39.5', 'second day','service_5');
insert into Choose_Service values('flat envelope', '84.3', 'longer','service_4');
insert into Choose_Service values('small box', '84.3', 'overnight','service_3');
insert into Choose_Service values('small box', '40', 'overnight','service_2');
insert into Choose_Service values('small box', '23', 'second day','service_3');
insert into Choose_Service values('small box', '55', 'second day','service_1');
insert into Choose_Service values('small box', '55', 'longer','service_1');
insert into Choose_Service values('larger box', '84.3', 'overnight','service_3');
insert into Choose_Service values('larger box', '24', 'overnight','service_3');
insert into Choose_Service values('larger box', '37', 'second day','service_3');
insert into Choose_Service values('larger box', '40', 'second day','service_3');
insert into Choose_Service values('larger box', '39.3', 'longer','service_3');
```

Package_Type, Weight, Timeliness에 해당하는 service를 알 수 있는 relation이다. 원래는 훨씬 더 많아서 Package와 one-to-many의 관계를 가질 수 있어야 되지만 여기서는 최소한으로 insert하였다.

4. Query

프로젝트 2에서 구현해야 될 query는 총 7개가 있다. Query를 설명하기 앞서서 가정한 것들 나열하겠다.

1. '사고가 발생한 Truck 1721은 2019-06-05 09:00:00에 출발하였다'

2. 'Truck 1721은 2019-06-05 11:45:00에 사고가 발생하였다'

3. '사고가 발생하면서 안에 있던 모든 Package들은 불타 없어졌다'

결국 C언어로 작성하여 실행이 되지만, 가독성을 위해 workbench에서 작성한 것을 가지고 설명을 진행하겠다.

4.1 Type-1.1

```
select Customer.Customer_ID, Customer.Customer_name
from Customer, Package as E, (select Package_ID as S
    from Move
    where Transportation_ID='1721'
    and Transportation_Device='Truck'
    and Departure_Time= '2019-06-05 09:00:00'
    and start_time < '2019-06-05 11:45:00'
    and end_time is NULL
)as T
where E.Package_ID = T.S
and Customer.Customer_ID = E.Customer_ID;
```

Query의 내용은 truck 1721이 사고가 발생하였을 때, 그 운송수단 안에 있던 Package에 해당하는 Customer들을 모두 출력하는 것이다. 우선 사고 당시에 타고 있던 Package들을 알기 위해서 'Move' relation을 살펴본다. Move에서 Transportation의 primary key인 (Transportation_ID, Transportation_Device, Departure_Time)을 사고가 발생한 1721 Truck으로 설정한다. 다음 사고 발생 시점 이전부터 탑승을 시작하였고, 도착을 아직 하지 않은 (NULL인) Package들을 골라내면, 사고 당시에 Truck 1721에 타고 있던 Package들을 식별할 수 있다. 다음 Package와 join하여 그 Package에 해당하는 Customer_ID를 구한 뒤, Customer와 join하여 그 customer들의 이름도 알아낼 수 있다. 그렇게 실행한 결과는 아래와 같다. 결과가 맞는지 확인하기 위해서는 위에 '3.4 Bills'를 insert하는 부분에서 알 수 있다. Move나 Package등에서 end_time이 NULL인 것들을 찾아도 된다.

```
----- TYPE 1-1 -----
** Find all customers who had a package on the truck at the time of the crash. **
-----
Customer_ID      Customer_name
-----
20161580         LeBron James
20161575         Ted Mosby
20161574         Seul Gee
20161578         Kim so Jin
```

4.2 Type-1.2

```
select Recipient.Recipient_ID, Recipient.Recipient_name
from Recipient, Package as E, (select Package_ID as S
    from Move
    where Transportation_ID='1721'
    and Transportation_Device='Truck'
    and Departure_Time= '2019-06-05 09:00:00'
    and start_time < '2019-06-05 11:45:00'
    and end_time is NULL
)as T
where E.Package_ID = T.S
and Recipient.Recipient_ID = E.Recipient_ID;
```

Type-1.2는 Type-1.1과 뽑아내야 되는 Package는 같지만 다른 점은 Customer가 아닌 Recipient를 골라내야 되는 것이다. 따라서 from에 Customer가 아닌 Recipient가 들어가고, Customer_name, Customer_ID가 아닌 Recipient_name, Recipient_ID를 출력한다. '3.4 Bills'를 insert하는 부분을 통해서 사고 당시의 truck에 있던 Package의 Recipient들이 누구인지 확인할 수 있다.

```
----- TYPE 1-2 -----
** Find all recipients who had a package on that truck at the time of the crash. **
-----
Recipient_ID      Recipient_name
-----
20161586          Stephen Curry
20161595          Ben Simmons
20161581          Sherlock Holms
20161595          Ben Simmons
```

4.3 Type-1.3

```
select Customer.Customer_ID, Customer_name, Recipient.Recipient_ID, Recipient_name, Arrival_Time
from Customer, Recipient, Package, (select Package_ID as S, max(end_time)
  from Move
 where Transportation_ID='1721'
 and Transportation_Device='Truck'
 and Departure_Time= '2019-06-05 09:00:00'
 and start_time < '2019-06-05 11:45:00'
 and end_time is not null
)as T
where Package.Package_ID = T.S
and Customer.Customer_ID = Package.Customer_ID
and Recipient.Recipient_ID = Package.Recipient_ID;
```

Type 1.3은 사고가 나기 직전에 성공한 Package에 대한 정보를 찾는 것이다. 우선 Transportation_ID, Transportation_Device, Departure_Time으로 'Truck 1721'을 골라내고 사고가 나기 이전에 출발하였고 도착시간이 NULL이 아닌 것을 골라낸다. 도착시간이 NULL이 아니라는 뜻은 도착시간이 있다는 뜻이고 배송에 성공했다는 뜻이다. 이런 Package들 중 max(end_time)을 가지는 Package를 골라낸다. 그 뒤 Customer, Recipient와 같이 join하여 해당하는 정보를 가져와서 아래와 같이 출력한다.

```
----- TYPE 1-3 -----
** Find the last successful delivery by that truck prior to the crash. **
-----
Customer_ID      Customer_name      Recipient_ID      Recipient_name      Arrival_Time
-----
20161576          Sheldon cooper      20161583          Bilbo Bagins          2019-06-05 11:30:00
```

4.4 Type-2

```
with T as (select Customer_ID as t_Customer_ID, count(Customer_ID) as order_num
  from Package
 where Year(order_time)=Year('19-01-01')#년도를 읽어와서 여기에 대입한다
 group by Customer_ID)
select Customer_ID, Customer_name
from T, Customer
where T.order_num = (select max(order_num) from T)
and t_Customer_ID = Customer_ID;
```

Type 2 query는 입력 받은 년도에 가장 많은 package를 배송 의뢰한 customer을 찾는다. 위의 예시는 19년도에 가장 많은 배송을 의뢰한 customer을 찾는 query이다. 우선 19년도에 택배를 배달한 기록이 있는 customer들을 Customer_ID와 총 몇번의 주문을 했는지를 나타내는 count(Customer_ID)를 order_num으로 하여 새로운 relation

T를 정의한다. 다음, T에서 order_num이 최대인 Customer_ID를 골라낸 뒤, Customer와 join하여 Customer_name까지 가져와서 같이 출력한다. 실제로 ODBC로 돌릴 때는 위의 Year('19-01-01') 부분에, 입력 받은 년도를 String형태로 대입하여 query를 실행하면 된다. Year()를 사용하면 뒤의 날짜는 무시하고 해당하는 timestamp의 년도끼리 비교를 하기 때문이다. 아래와 같이 'sprintf'를 사용하였다.

```
sprintf(query, "With T as (select Customer_ID as t_Customer_ID, sum(Cost) as total_Cost from Package, Bills where Year(order_time)=Year('%d-01-01') and Bills.Package_Id = Package.Package_ID group by Customer_ID) select Customer_ID, Customer.Customer_name from T, Customer where T.total_Cost=(select max(total_Cost) from T) and t_Customer_ID = Customer_ID; ", year);
```

이를 실행하면 다음과 같은 결과를 얻을 수 있다.

```
---- TYPE II ----
** Find the customer who has shipped the most packages in certain year **
Which year? 2019
Customer_ID      Customer_name
-----
20161576         Sheldon cooper
```

만약 해당하는 년도에 저장된 정보가 없다면, 즉 해당하는 년도에 package를 보낸 customer가 없다면 오류를 출력하고 다시 년도를 받는 부분으로 돌아가도록 설정하였다. 프로젝트에서 사용한 insert는 2017~2019사이의 정보들 로만 이루어져 있으므로 그 외의 년도를 입력하면 아래와 같은 오류 문구가 나올 것이다.

```
** Find the customer who has shipped the most packages in certain year **
Which year? 2015
No Record in the certain year
```

4.5 Type-3

```
with T as (select Customer_ID as t_Customer_ID, sum(Cost) as total_Cost
from Package, Bills
where Year(order_time)=Year('19-01-01')#년도를 읽어와서 여기에 대입한다
and Bills.Package_Id = Package.Package_ID
group by Customer_ID)
select Customer_ID, Customer.Customer_name
from T, Customer
where T.total_Cost=(select max(total_Cost) from T)
and t_Customer_ID = Customer_ID;
```

Type-3는 특정 년도에 가장 package에 돈을 많이 사용한 customer를 선택한다. Type-2와 특정 년도를 선택하는 방식은 같지만 'Bills'가 과정에 포함된다는 차이점이 있다. 해당하는 package의 cost들의 sum을 Customer_ID로 grouping한다. 이렇게 하여 새로운 relation T를 만든다. T에는 Customer_ID, 그리고 그 customer가 해당 년도에 사용한 총 금액을 저장하는 total_Cost가 저장되어 있다. 이제 그 T와 Customer를 join하여 total_Cost가 max인 customer를 색출한다. 19년도에 가장 많은 돈을 사용한 고객은 아래와 같이 'Kim so Jin'이다.

```
---- TYPE III ----
** Find the customer who has spent the most money on shipping in the past certain year **
Which year? 2019
Customer_ID      Customer_name
-----
20161578         Kim so Jin
```

Type-2와 마찬가지로 해당하는 년도에 저장된 data가 없다면 오류를 출력하고 다른 년도를 입력 받는다.

```
** Find the customer who has spent the most money on shipping in the past certain year **
Which year? 2015
No Record in the certain year
```

4.6 Type-4

```
select Package_ID, Customer.Customer_ID, Customer.Customer_name, Recipient.Recipient_ID, Recipient.Recipient_name, Timeliness
from Package, Recipient, Customer
where ((Timeliness = 'overnight' and timestampdiff(hour,Order_Time,Arrival_time) > 24)
OR (Timeliness = 'second day' and timestampdiff(hour,Order_Time,Arrival_time) > 48))
and customer.customer_id = package.customer_id
and recipient.recipient_id = package.recipient_id;
```

Type-4는 timeliness안에 도착하지 못한 package들을 찾아내는 query이다. 명세서의 조건에 따라서 timeliness는 'overnight', 'second day', 'longer'로 총 3가지 경우가 있다. 'longer'의 경우에는 제한 날짜가 없기 때문에 고려할 필요가 없다. 'overnight'의 경우 주문 시간 기준 24시간 이내에 도착하는 것을 기준으로 삼았고, 'second day'의 경우에는 주문시간 기준 48시간 이내에 도착하는 것을 기준으로 삼았다. 두개의 timestamp 자료형을 비교할 수 있는 timestampdiff()를 사용하여 order_time과 arrival_time을 'hour'단위로 비교하였다. Recipient와 Customer를 join하여 Customer_name, Recipient_name도 같이 출력하였다. Type-4는 따로 입력을 받지 않기 때문에 바로 실행이 되고 실행이 완료된 이후 다시 Query Type을 설정하는 보기로 돌아가게 작성하였다.

---- TYPE IV ----					
** Find those packages that were not delivered within the promised time **					
Package_ID	Customer_ID	Customer_name	Recipient_ID	Recipient_name	Timeliness
17-0000001	20161572	Jo Eun Gee	20161585	Michael Jordan	second day
17-0000003	20161580	LeBron James	20161590	Kevin Love	second day
17-0000004	20161580	LeBron James	20161593	Deric Rose	overnight
18-0000001	20161570	Son	20161590	Kevin Love	overnight
18-0000004	20161579	Anna Kendrick	20161592	Demar Derozan	overnight

4.7 Type-5

Type-5는 총 3종류의 bill을 출력하였다.

1. simple bill
2. An itemize billing listing each individual shipment and the charges for it
3. A bill listing charges by type of service

Type-5에서의 예시는 Customer_ID가 '20161579'인 고객의 '2018년 2월' bill을 기준으로 작성하였다. 이 부분은 C파일의 query에선 sprintf를 사용하여 입력 받은 ID와 년, 월을 기준으로 돌아가도록 만들었다. 구현한 Customer relation의 primary key가 name이 아니라 ID임으로 입력을 name으로 받지 않고 ID로 받았다. 또한 모든 bill은 도착한 시간 기준이 아닌 주문한 시간을 기준으로 작성하였다.

→우선 simple bill을 살펴보면 다음과 같다.

```
select package.Customer_ID, sum(bills.cost)as Total_Cost
from package, bills
where package.package_id=bills.package_id
and package.customer_ID='20161579' #찾고 싶은 customer ID
and year(package.order_time) = year('2018-01-01') #년도
and month(package.order_time)=month('2018-02-01'); #달
```

해당하는 Customer_ID와 Year, Month를 설정하여 추출한다. Simple bill에서는 총 사용한 금액만 필요하므로 cost의 sum을 구하여 Total_Cost라는 새로운 항목으로 만들었다. 이때 year()와 Month()에 들어가는 timestamp는 해당하는 항목만 설정해 주면 된다. 예를 들어서 year()안에 들어 있는 timestamp는 년도만 보고 나머지 월과 일을 고려하지 않는다. 따라서 year(package.order_time)=year('2018-01-01')을 하면 2018년에 주문한 package들을 선택하게 된다.

→ 다음 itemized bill을 살펴보자.

```
select package.Package_ID, Cost, Prepaid, Package_type, Timeliness, Hazardous, international
from Package, Bills
where Package.package_ID=Bills.package_Id
and package.customer_ID='20161579'           #찾고 싶은 customer ID
and year(package.order_time) = year('2018-01-01') #년도
and month(package.order_time)=month('2000-02-01'); #달
```

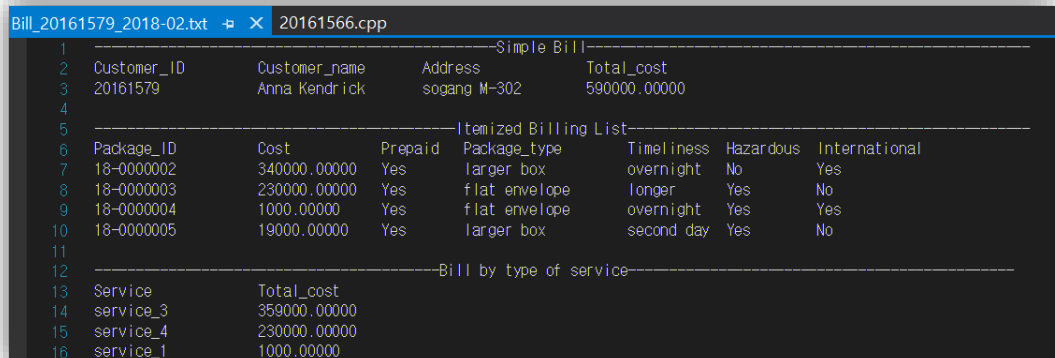
Itemized bill은 package별로 가격, type, timeliness등의 package에 대한 정보 또한 같이 출력한 bill이다. 따라서 앞의 simple bill과 같이 특정 package를 색출해낸 다음, bill과 join하여 해당하는 package의 특징들을 같이 출력하면 된다.

→ 마지막으로 bill listing charges by type of service을 살펴보자.

```
select Service, Sum(cost)as Total_cost
from Package, Bills, Choose_Service
where Package.package_ID=Bills.package_Id
and package.customer_ID='20161579'
and year(package.order_time) = year('2018-01-01')
and month(package.order_time)=month('2000-02-01')
and Package.Package_type=Choose_Service.Package_type
and Package.Weight = Choose_Service.Weight
and Package.Timeliness = Choose_Service.Timeliness group by Service;
```

특정 고객이 의뢰한 package들을 service별로 묶어서 가격을 표시한 것이 'bill listing charges by type of service'이다. 우선 앞의 bill들과 마찬가지로 해당하는 ID와 날짜에 맞는 package들을 골라낸다. 다음 Service를 알기 위해 Choose_Service와 join을 하는데, 이때 Choose_Service의 primary key가 type, weight, timeliness라서 모두 다 조건에 포함해야 된다. 다음 Service로 grouping을 하면서 같은 service에 있는 cost들을 sum해주어 Total_cost라는 이름으로 저장하였다. 이렇게 하면 Service별로 총 얼마의 금액을 사용하였는지 알 수 있다.

이 모든 bill들을 하나의 출력 txt파일에 출력하였고, 결과는 다음과 같다.



```
Bill_20161579_2018-02.txt 20161566.cpp
```

-----Simple Bill-----							
	Customer_ID	Customer_name	Address	Total_cost			
1	20161579	Anna Kendrick	sogang M-302	590000.00000			
-----Itemized Billing List-----							
	Package_ID	Cost	Prepaid	Package_type	Timeliness	Hazardous	International
6	18-0000002	340000.00000	Yes	larger box	overnight	No	Yes
7	18-0000003	230000.00000	Yes	flat envelope	longer	Yes	No
8	18-0000004	1000.00000	Yes	flat envelope	overnight	Yes	Yes
9	18-0000005	19000.00000	Yes	larger box	second day	Yes	No
-----Bill by type of service-----							
	Service	Total_cost					
13	service_3	359000.00000					
14	service_4	230000.00000					
15	service_1	1000.00000					

Bill의 이름은 Customer_ID, 그리고 지정한 날짜로 하였다.

처음 입력을 받는 부분에서, 만약 해당하는 ID의 Customer가 존재하지 않는 경우 혹

은 해당하는 Customer에 지정한 달에 주문 기록이 없는 경우에는 각각에 맞는 오류 문구를 출력하고 다시 처음부터(Customer_ID 입력) 진행을 하게 하였다. 또한 마지막 탈출 조건으로는 Customer_ID를 입력 받는 란에 0을 입력하면 Query Type을 결정하는 화면으로 돌아가게 설정하였다.

```

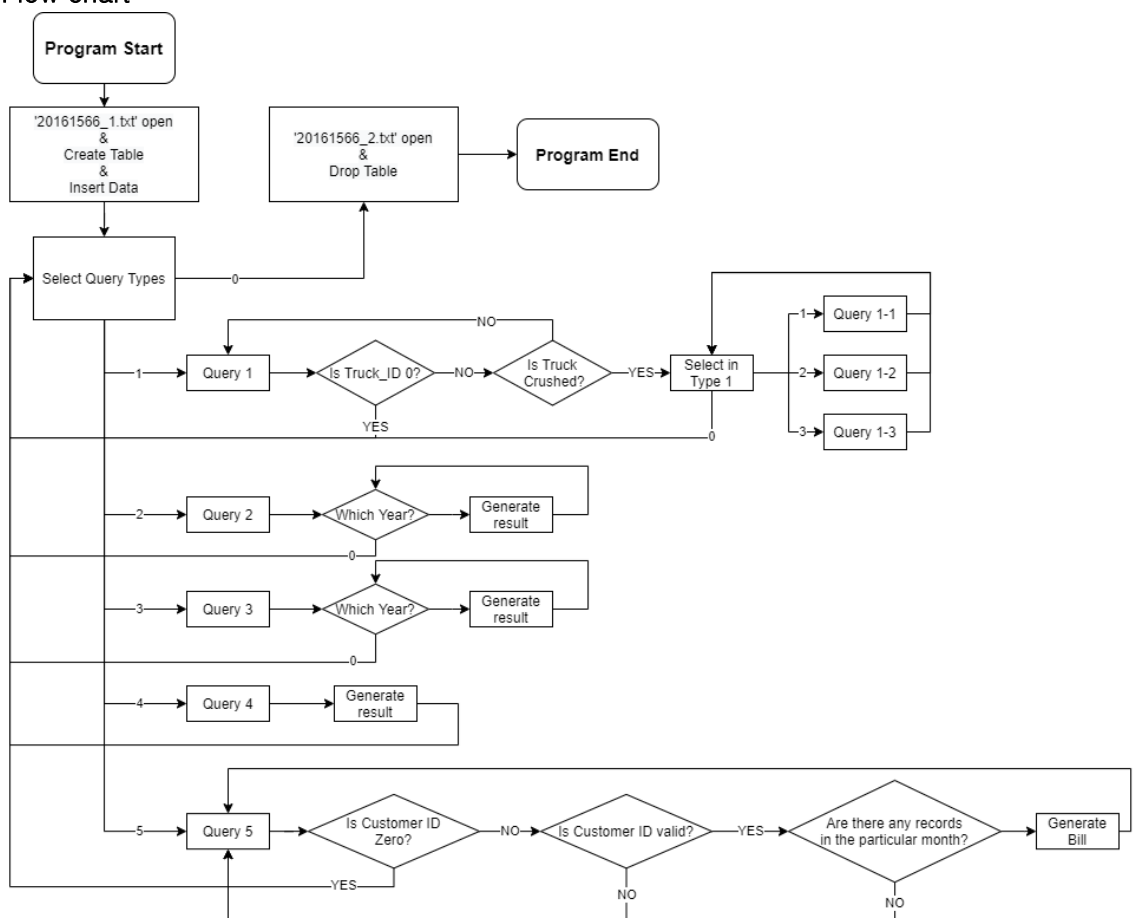
** Generate the bill for each customer for the past certain month **
Customer ID: 20152322
There is no such Customer

** Generate the bill for each customer for the past certain month **
Customer ID: 20161579
Which month(YYYY-MM)? 2015-03
There is no such Record in that month

```

5. C-Code

5.1 Flow chart



C언어로 작성한 코드는 위와 같은 Flow chart를 따라간다. 처음 '20161566_1.txt'를 연다. 이 안에는 CRUD의 내용들이 담겨있으며, 한줄씩 읽어서 Create table과 insert into를 실행한다. 다음 Query Type을 정하는 while문에 들어가게 된다. 0을 입력 받기 전까지 이 while문 안에서 계속 입력을 받게 된다. 입력 받는 숫자에 따라서 Query가 선택된다.

→Query 1의 경우 처음 Truck의 ID를 입력 받는다. 이때, 입력 받은 ID가 0이라면 'Select Query Types'로 돌아간다. 0이외의 수를 입력 받았다면, 그 입력 받은 ID의 Truck이 부쉬진 truck(1721)인지 확인한다. 아니라면 다시 ID를 입력 받는다. 만약 1721 truck이 맞다면 세부 type을 정하게 된다. 이때 0을 입력하면 'Select Query Types'로 돌아가게 된다.

→Query 2의 경우 처음에 년도를 입력 받는다. 이때 0을 받으면 다시 'Select Query

Types'로 돌아가고 그 외의 숫자를 입력 받으면 해당하는 년도로 query를 돌린다. 단, 해당하는 년도에 저장된 정보가 없다면 다시 년도를 입력 받는다.

→Query 3은 Query2와 구조가 완전히 똑같다.

→Query 4는 따로 입력을 받지 않기 때문에 결과를 출력하고 다시 'Select Query Types'로 돌아간다.

→Query 5는 총 3번의 입력을 확인하는 단계를 거친다. 우선 'Select Query Types'로의 탈출 조건은 처음 Customer ID로 0을 입력 받았을 시이다. 0이 아닌 수를 ID로 받았다면 그 ID가 valid한지, 즉 그런 Customer_ID가 존재하는지 확인하고 없다면 다시 Customer_ID를 입력 받는다. 만약 있다면 년도와 해당하는 달을 입력 받는데, 이때 앞에서 입력 받은 customer에 해당하는 달에 저장된 정보가 없다면 Customer_ID를 입력 받는 부분으로 다시 돌아간다.

'Select Query Types'에서 0을 입력 받은 이후 '20161566_2.txt'를 열어 drop table query들을 실행시킨 뒤 프로그램을 종료한다.

5.2 '20161566.cpp'

<사진 생략>

입력 받은 값을 query에 적용하는 부분을 잠깐 설명하자면 다음과 같다.

```
printf("Customer ID: ");
scanf("%s", ID);
if (strcmp(ID, "0") == 0) //Customer_ID로 0이 입력된 경우 Query Type select 로 돌아간다.
    break;

sprintf(query, "select * from Customer where Customer_ID=%s", ID);
mysql_query(connection, query);
sql_result = mysql_store_result(connection);
if ((sql_row = mysql_fetch_row(sql_result)) == NULL) { //ID가 존재하지 않는다면 다시 ID를 입력 받는다.
    printf("There is no such Customer\n\n");
    continue;
}
```

위의 코드는 Query 5에서 Customer_ID를 입력 받은 이후, 그 Customer_ID가 존재하는 ID인지 확인하는 query이다. 빨간색 네모 안의 부분이 query를 생성하는 부분이다. 완성된 query에 ID가 들어갈 자리만 %s를 넣고 뒤에 그 %s에 들어갈 string을 입력함으로써 query라는 문자열에 실질적으로 실행시킬 query를 입력하게 된다. 위의 코드에서 입력을 받아 query를 동작하는 모든 query들은 이러한 방식을 사용하였다. 입력을 받지 않는 query의 경우에는 처음에 정해진 query를 query변수에 넣고 계속해서 살펴보면, "mysql_query", "mysql_store_result", "mysql_fetch_row"등의 mysql에 관련된 함수들이 나온다. 각각을 간단하게 설명하면 다음과 같다.

→mysql_query

```
int mysql_query(MYSQL *mysql, const char *stmt_str)
```

위와 같은 형식으로 사용이 된다. Stmt_str에 저장된 string을 query로 취급하여 실행시킨다. 보통 하나의 query를 실행시키는데 사용되며, 성공하면 0을 실패하면 0이외의 값을 반환한다.

→mysql_store_result

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Mysql_query를 invoke한 후 꼭 mysql_store_result를 사용하여 결과를 저장해야 된다. 이후에 결과 값 사용을 완료하면 mysql_free_result를 사용하여 결과 값을 삭제한다.

→mysql_fetch_row

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

Mysql_store_result를 사용한 이후 저장된 row가 하나도 없다면 NULL을 반환하고 있다면 해당하는 row를 반환한다. 위에서는 while문을 통하여 각 row에 access하게 된다.

다음은 query 3에 해당하는 코드이다. 대부분 query가 같은 흐름으로 흘러가기 때문에 대표적으로 query 3을 예시로 설명하겠다.

```
case 3: //query 3번 선택
printf("---- TYPE III ----\n");
while (1) {
    int year = 0;
    printf("*** Find the customer who has spent the most money on shipping in the past certain year **\n");
    printf("Which year? ");
    scanf("%d", &year);
    if (year == 0) //year가 0이면 query선택으로 돌아간다.
        break;
    sprintf(query, "with T as (select Customer_ID as t_Customer_ID, sum(Cost) as total_Cost from Package, Bills where Year = %d)", year);
    mysql_query(connection, query);
    sql_result = mysql_store_result(connection); //해당 year에 아무런 기록이 없는 경우
    if ((sql_row = mysql_fetch_row(sql_result)) == NULL) {
        printf("No Record in the certain year\n\n");
        continue;
    }
    mysql_query(connection, query); //해당 year에 기록이 있는 경우, query를 돌려서 결과를 저장한다.
    sql_result = mysql_store_result(connection);
    printf("Customer_ID      Customer_name\n");
    printf("-----\n");
    while ((sql_row = mysql_fetch_row(sql_result)) != NULL) //결과를 출력
    {
        printf("%-20s%-20s\n", sql_row[0], sql_row[1]);
    }
    mysql_free_result(sql_result);
    printf("\n");
}
break;
```

우선 query에 들어오면 어떤 type인지 출력을 하고 while문에 들어가게 된다. 다음 입 출력 과정을 지나서 sprintf로 query를 작성하게 된다. 앞에서 설명한 mysql이 함수들을 통해서 결과 row를 가지고 와서 sql_row에 저장한 뒤, sql_row[n]의 형태로 각각의 값들을 순서대로 출력한다. 이때 출력은 query에 따라서 형식, stdout이나 txt파일이나 등이 결정이 된다. 마지막에 mysql_free_result를 해주는 것도 볼 수 있다.

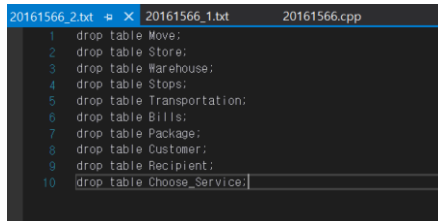
5.3 '20161566_1.txt'

```
20161566_1.txt 20161566.cpp
1 create table Choose_Service(Package_type varchar(20) check (Package_type='flat envelope' OR Package_type='small box',
2 create table Customer(Customer_ID varchar(20), Customer_name varchar(20), Address varchar(20), Contact numeric(15,0), Pa
3 create table Recipient(Recipient_ID varchar(20), Recipient_name varchar(20), Address varchar(20), Contact numeric(15,0), pr
4 create table Package(Package_ID varchar(20), Customer_ID varchar(20), Recipient_ID varchar(20), Package_type varchar(
5 create table Bills(Package_ID varchar(20), Prepaid varchar(5) check(Prepaid='Yes' OR Prepaid='No'), Cost numeric(15,5), pr
6 create table Warehouse(Warehouse_ID varchar(20), Location varchar(20), Contact numeric(12,0), primary key(Warehouse_ID))
7 create table Store(Package_ID varchar(20), Warehouse_ID varchar(20), Start_Time timestamp, End_Time timestamp, primary ke
8 create table Transportation(Transportation_ID varchar(20), Departure_Time timestamp, Transportation_Device varchar(10),
9 create table Move(Package_ID varchar(20), Transportation_ID varchar(20), Departure_Time timestamp, Transportation_Dev
10 create table Stops(Location varchar(20), Start_Time timestamp, Transportation_ID varchar(20), Departure_Time timestamp, Tr
11 insert into Choose_Service values('flat envelope', '37.5', 'overnight', 'service_1');
12 insert into Choose_Service values('flat envelope', '90', 'overnight', 'service_1');
13 insert into Choose_Service values('flat envelope', '39', 'second day', 'service_1');
14 insert into Choose_Service values('flat envelope', '39.5', 'second day', 'service_5');
15 insert into Choose_Service values('flat envelope', '84.3', 'longer', 'service_4');
16 insert into Choose_Service values('small box', '84.3', 'overnight', 'service_3');
17 insert into Choose_Service values('small box', '40', 'overnight', 'service_2');
18 insert into Choose_Service values('small box', '23', 'second day', 'service_3');
19 insert into Choose_Service values('small box', '55', 'second day', 'service_1');
20 insert into Choose_Service values('small box', '55', 'longer', 'service_1');
21 insert into Choose_Service values('larger box', '84.3', 'overnight', 'service_3');
22 insert into Choose_Service values('larger box', '24', 'overnight', 'service_3');
23 insert into Choose_Service values('larger box', '37', 'second day', 'service_3');
24 insert into Choose_Service values('larger box', '40', 'second day', 'service_3');
25 insert into Choose_Service values('larger box', '39.3', 'longer', 'service_3');
26 insert into Customer values('20161566', 'Kwon Hyung Joon', 'sogang K-301', '01023454321', 'Infrequent', NULL);
27 insert into Customer values('20161567', 'Chun He Jin', 'sogang R-912', '01022020991', 'Infrequent', NULL);
28 insert into Customer values('20161568', 'Jung Jin One', 'sogang R-914', '01046933324', 'Infrequent', NULL);
29 insert into Customer values('20161569', 'Kang Min Suk', 'sogang RA-993', '01030204039', 'Infrequent', NULL);
30 insert into Customer values('20161570', 'Son', 'sogang M-101', '01012340938', 'Infrequent', NULL);
31 insert into Customer values('20161571', 'Kim Su Hyun', 'sogang R-123', '01030491009', 'Infrequent', NULL);
32 insert into Customer values('20161572', 'Jo Eun Gee', 'sogang D-303', '01010930023', 'Infrequent', NULL);
33 insert into Customer values('20161573', 'Park Seung Joo', 'sogang D-301', '01057684955', 'Infrequent', NULL);
34 insert into Customer values('20161574', 'Saul Gee', 'sogang K-401', '01099807783', 'frequent', '1034004924');
35 insert into Customer values('20161575', 'Ted Mosby', 'sogang AS-202', '01029384820', 'frequent', '120341435234');
36 insert into Customer values('20161576', 'Sheldon cooper', 'sogang R-505', '01009877899', 'frequent', '120349120348');
37 insert into Customer values('20161577', 'Rajesh Cothropolis', 'sogang J-101', '01078976788', 'frequent', '09839573023');
38 insert into Customer values('20161578', 'Kim Se Jin', 'sogang K-303', '01039489947', 'frequent', '34219395992');
39 insert into Customer values('20161579', 'Anna Kendrick', 'sogang W-302', '01039220893', 'frequent', '0093927492701');
40 insert into Customer values('20161580', 'LeBron James', 'sogang R-912', '01000098889', 'frequent', '14148582393');
```

위에서 설명한 Create table, Insert into를 각각 한 줄로 정리하여 입력하였다. File을 열어서 한 줄씩 string으로 읽어온 이후 바로 query로 사용하기 때문에 빈 줄은 허용이 안 되었다. (위의 사진은 일부만 포함한 것) 여기서 주의할 것은 create table의 순서인데, FK를 포함하고 있는 table은 해당 FK가 선언된 table이 생성되고 난 이후에 create가 가능하였다. 다음은 이를 어길 시 발생하는 오류를 work bench에서 가져온 것이다.

Error Code: 1824. Failed to open the referenced table 'recipient'

5.4 '20161566_2.txt'



```
20161566_2.txt 20161566_1.txt 20161566.cpp
1 drop table Move;
2 drop table Store;
3 drop table Warehouse;
4 drop table Stops;
5 drop table Transportation;
6 drop table Bills;
7 drop table Package;
8 drop table Customer;
9 drop table Recipient;
10 drop table Choose_Service;
```

프로그램이 종료될 때 처음에 생성한 table들을 drop하는 과정이다. Delete는 table의 tuple (row)를 삭제하는 과정이고, drop은 table 자체를 삭제하는 과정이다. 따라서 delete는 생략하고 drop하는 부분만 넣었다. 이때도 순서가 중요한데, 만약 FK가 다른 table에 포함되어 있는 table을 먼저 삭제하게 되면 foreign key constraint로 인해서 다음과 같은 오류가 발생한다.

Error Code: 3730. Cannot drop table 'customer' referenced by a foreign key constraint 'package_ibfk_3' on table 'package'.

감사합니다!