[Database System]

Project 2

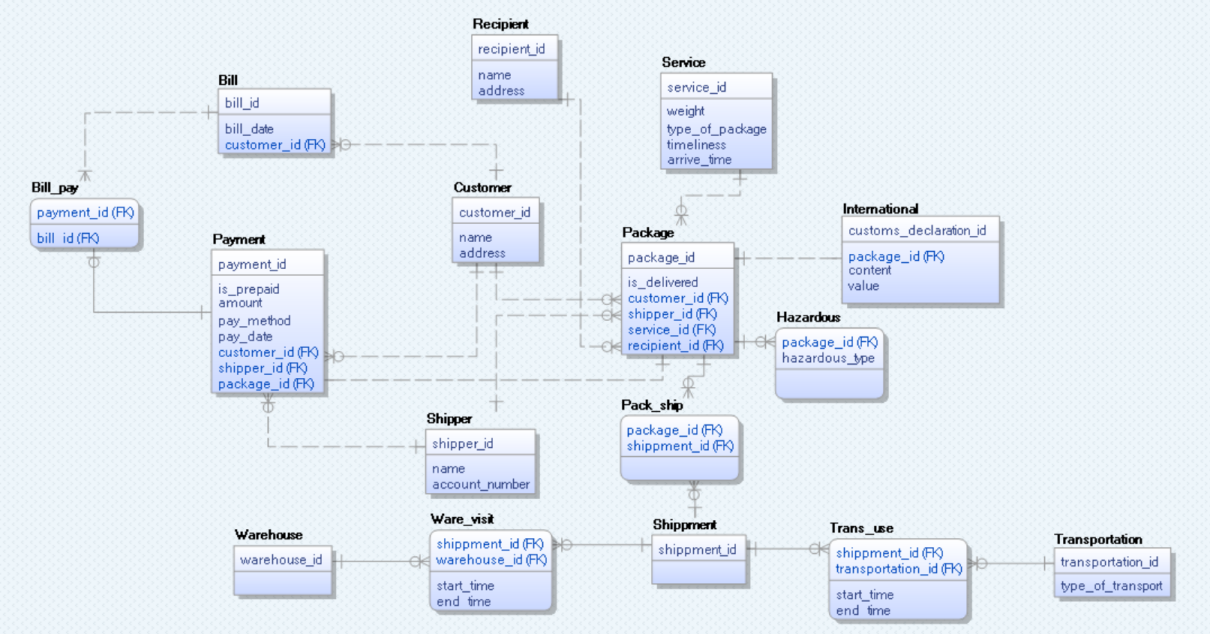
담당 교수 : 정성원 교수님

전공 : 컴퓨터공학

학번 : 20191617

이름 : 이규호

1. BCNF Decomposition



BCNF Decomposition을 하기에 앞서, project1에서 잘못 이해하거나 명확하지 않은 부분을 변경하였다.

1) Transportation entity set의 type_of_transprot를 primary key에서 제외하여 transportation의 type이 다르면 다른 id를 갖는 것으로 구현했다.

2) Trans_use, Ware_visit으로 정확한 도착 여부를 확인하기 모호하여 Service에 실제 도착 시간인 arrive_time 추가했다.

3) Bill entity set에서 bill_month를 bill_date로 바꾸었다.

4) Bill entity set에서 bill_type을 구분하여 제공하는 것은 data의 수가 늘어남에 따라 크기가 커질 것을 고려하여 bill_type을 삭제 했다.

5) payment에 해당하는 package를 찾을 수 없는 문제가 있어 payment - package 사이의 relation을 추가하였다.

***BCNF decomposition**

모든 entity와 relationship에 대하여 BCNF dependency를 만족해야한다고 명시되어 있기 때문에, 각각의 entity와 relationship에 대하여 super key인지 확인하고 super key 아닌 attribute set a 에 대하여 a->b 를 만족하는 non-trivial한 FD의 존재 유무를 확인하였다.

BCNF simplified test를 통해 먼저 International table을 우선적으로 제시하고 이후 다른 모든 entity set 에 대하여 BCNF 여부를 확인하였다.

International : International table은 아래와 같이 2개의 FD를 가진다.

1. customs_declaration_id -> package_id, content, value

2. package_id -> content, value

1번 FD의 경우 customs_declaration_id는 primary key이기 때문에 당연하게도 super key 역할을 하기 때문에 BCNF의 조건을 만족한다.

2번 FD의 경우 Package table과 International table은 일대일 관계이기 때문에 package_id 자체로 super key이자 candidate key의 역할을 한다. 따라서 2번 FD 또한 BCNF의 조건을 만족한다.

따라서 International entity set은 BCNF dependency를 만족한다고 할 수 있다.

이어서 다른 모든 entity와 relationship에 대한 BCNF 여부이다.

Customer : primary key인 customer_id 외에는 다른 FD가 존재하지 않는다고 판단하였다. name의 경우 동명이인이 존재하여 name이 같아도 다른 address를 가질 수 있다고 판단하여 FD가 존재하지 않는다고 판단하였고, 결과적으로 Customer는 BCNF dependency를 만족한다.

Recipient : primary key인 customer_id 외에는 다른 FD가 존재하지 않는다고 판단하였다. customer와 마찬가지로 name의 경우 동명이인이 존재하여 name이 같아도 다른 address를 가질 수 있다고 판단하여 FD가 존재하지 않는다고 판단하였고, 결과적으로 Recipient는 BCNF dependency를 만족한다.

Shipper : primary key인 shipper_id 외에는 다른 FD가 존재하지 않는다고 판단하였다. customer와 마찬가지로 name의 경우 동명이인이 존재하여 name이 같아도 다른 account_number를 가질 수 있다고 판단하여 FD가 존재하지 않는다고 판단하였고, 결과적으로 Shipper는 BCNF dependency를 만족한다.

Service : primary key인 service_id 외의 다른 attribute set에 대하여 다른 FD가 존재 하지 않는다고 판단하여 Service는 BCNF dependency를 만족한다.

Package : primary key인 package_id 외의 다른 FD가 존재하지 않는다고 판단하였다. 따라서 Package는 BCNF dependency를 만족한다.

Hazardous : Package와 Hazardous는 일대다 관계이기 때문에 package_id -> hazardous_type의 FD는 존재하지 않는다. 따라서 Hazardous entity set에는 FD가 존재하지 않으므로 BCNF dependency를 만족한다.

Pack_ship : 마찬가지로 FD가 존재하지 않기 때문에 BCNF dependency를 만족한다.

Shippment : Shippment 또한 FD가 존재하지 않아 BCNF dependency를 만족한다.

Warehouse : Warehouse도 FD가 존재하지 않아 BCNF dependency를 만족한다.

Transportation : primary key인 transportation_id 외의 FD는 존재하지 않기 때문에 BCNF dependency를 만

족한다.

Ware_visit : primary key인 (shippment_id, warehouse_id) 외의 FD는 존재하지 않으므로 BCNF dependency를 만족한다.

Trans_use : primary key인 (shippment_id, transportation_id) 외의 FD는 존재하지 않으므로 BCNF dependency를 만족한다.

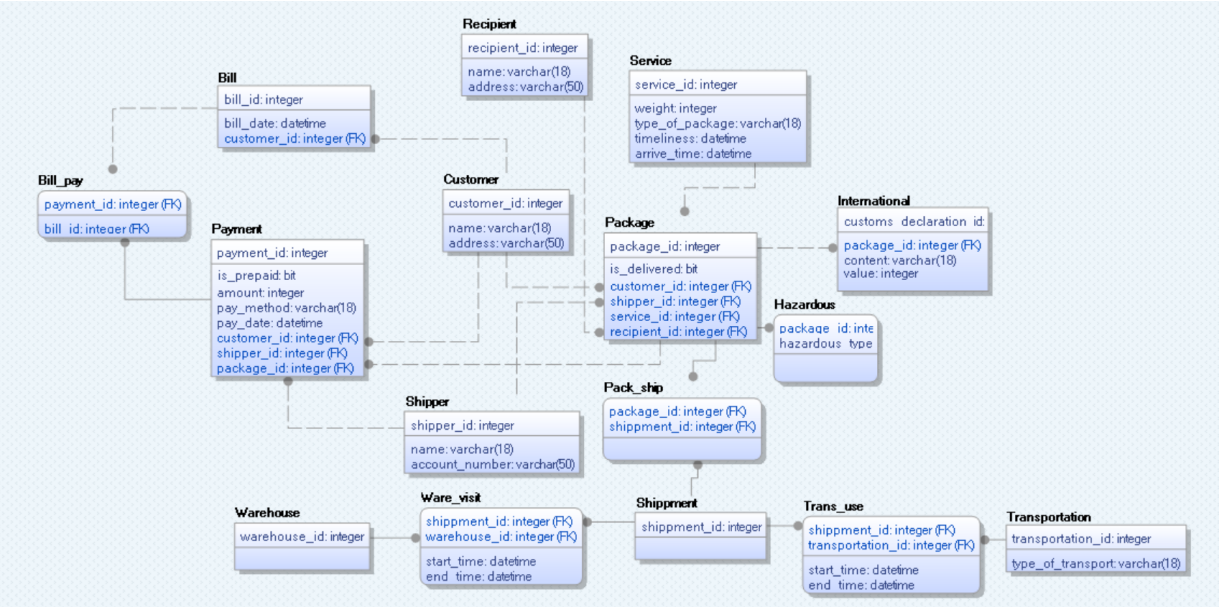
Payment : 먼저 primary key에 해당하는 payment_id가 a가 되는 FD가 있다. 그 외의 non-trivial한 FD가 있는지 확인하였을 때, bill로 shipper에게 pay를 하는 customer 또한 선불해야하는 특수한 경우에는 credit card로 돈을 지불해야하기 때문에, is_prepaid -> pay_method 인 FD는 존재하지 않는다고 판단하였다. 따라서 customer_id -> pay_method 인 FD 또한 존재하지 않는다고 판단하였다. 또한 Package와 Payment는 일대일 관계이기 때문에 package_id 자체로 super key이자 후보키 역할을 한다. 따라서 존재하는 두 FD에 대하여 BCNF를 만족하기 때문에 Payment는 BCNF dependency를 만족한다.

Bill : primary key인 bill_id 외의 FD가 존재하지 않기 때문에 Bill은 BCNF dependency를 만족한다.

Bill_pay : primary key인 payment_id 외의 FD가 존재하지 않기 때문에 Bill_pay는 BCNF dependency를 만족한다.

2. Physical Schema Diagram

각각의 relation에 대하여 data type, domain, constraints, relation type, allowing nulls 등을 고려하여 physical schema diagram을 구현하였다.



Package

- package_id (PK) : package의 id를 저장하며 정수 값이 들어간다. PK이기 때문에 not null로 설정했다. Package entity set은 package의 배달 완료 여부, package에 해당하는 customer, shipper, service, recipient 등의 정보를 저장한다.

- is_delivered : 배달 완료 여부를 저장하기 위해 boolean type으로 설정하였고, physical로 변환 시 bit라고 표기되었다. 배달 완료 여부이기 때문에 not null로 설정했다.

- customer_id (FK) : customer의 id를 저장하며 정수 값이 들어간다. FK이기 때문에 not null로 설정했다.

- shipper_id (FK) : shipper의 id를 저장하며 정수 값이 들어간다. FK이기 때문에 not null로 설정했다.

- service_id (FK) : service_id의 id를 저장하며 정수 값이 들어간다. FK이기 때문에 not null로 설정했다.

- recipient_id (FK) : recipient의 id를 저장하며 정수 값이 들어간다. FK이기 때문에 not null로 설정했다.

Customer

- customer_id (PK) : customer의 id를 저장하며 정수 값이 들어간다. PK이기 때문에 not null로 설정했다. Customer entity set은 customer의 이름, 주소를 저장한다.

- name : customer의 이름을 저장하기 위해 name의 size를 고려하여 VARCHAR(18)로 data type을 설정하였고, not null로 설정했다.

- address : customer의 주소를 저장하기 위해 address의 size를 고려하여 VARCHAR(50) 으로 data type을 설정하였다. address 또한 not null로 설정하여 위의 name과 address를 무조건 가지도록 하였다.

Shipper

- shipper_id (PK) : shipper 의 id를 저장하며 정수 값이 들어간다. PK이기 때문에 not null로 설정했다. Shipper entity set은 shipper의 이름과 계좌번호 정보를 저장한다.

- name : shipper의 이름을 저장하기 위해 name의 size를 고려하여 마찬가지로 VARCHAR(18)로 설정하였고, not null로 설정했다.

- account_number : 'SH789012', 'WR567890' 와 같은 계좌번호가 들어온다고 가정하였고, 길이가 긴 계좌번호가 들어오는 경우가 존재한다고 판단되어 VARCHAR(50)으로 data type을 설정하였다. 계좌번호가 있어야 계약을 맺은 customer가 돈을 지불할 수 있기 때문에 not null 로 설정하였다.

Service

- service_id (PK) : service 의 id를 저장하며 정수 값이 들어간다. PK이기 때문에 not null로 설정했다. service id로 unique하게 식별되는 service는 무게, 패키지의 타입, 예상 도착 시간 등에 따라 service의 종류가 달라진다.

- weight : package의 무게를 저장하기 위해 integer type으로 설정하고, not null로 설정했다.

- type_of_package : ‘envelope’, ‘small box’, ‘large box’ 등과 같은 package의 type을 저장하기 위해 VARCHAR(18)로 data type을 설정하고, not null로 설정하였다.

- timeliness : 예상 도착 시간으로, 이를 저장하기 위해 datetime으로 data type을 설정하고, not null로 설정하였다. '2023-06-02 10:00:00'와 같이 저장된다.

- arrive_time : package의 실제 도착 시간으로, timeliness와 마찬가지로 datetime으로 설정하고, 아직 도착하지 못한 package도 존재할 수 있기 때문에 null로 설정하여 null을 허용하였다.

Recipient

- recipient_id (PK) : recipient의 id를 저장하며 정수 값이 들어간다. PK이기 때문에 not null로 설정했다. Recipient entity set은 수령인의 이름과 주소를 저장한다.

- name : recipient 의 이름을 저장하기 위해 name의 size를 고려하여 마찬가지로 VARCHAR(18)로 설정하였고, not null로 설정했다.

- address : recipient의 주소를 저장하기 위해 address의 size를 고려하여 VARCHAR(50) 으로 data type을 설정하였다. address 또한 not null로 설정하여 위의 name과 address를 무조건 가지도록 하였다. recipient의 address는 shippment의 목적지이기도 하다.

International

- customs_declaration_id (PK) : 세관신고가 필요한 국제 배송의 경우에 package에 해당하는 세관신고의 id를 저장하고, integer로 data type을 설정하였다. primary key 이기 때문에 not null로 설정했다. International entity set은 국제배송이고 세관신고가 필요한 경우에 대해서 세관신고가 필요한 package, content, value 등의 정보를 저장한다.

- package_id (FK) : foreign key로 package_id를 가져와 세관신고에 해당하는 package_id를 attribute로 가지도록 하였다. package_id는 integer type이며, not null이다.

- content : 세관신고가 필요한 package의 경우 해당 package의 내용물을 알 필요가 있다. 따라서 content를 통해 이를 명시하고, 내용물 이름의 size를 고려하여 VARCHAR(18)로 설정하였다. 이는 not null로 설정했다.

- value : 또한 세관신고가 필요한 package의 경우 내용물의 가치 또한 database에서 알 필요가 있다. 따라서 value를 integer data type으로 설정하여 그 가치를 저장하도록 하였다. 이 database에서 돈의 단위는 달러로 들어온다고 가정하였다. 이 또한 not null로 설정했다.

Hazardous

- package_id (PK, FK) : 배송회사는 hazardous한 package에 대한 정보도 알고 있을 필요가 있다. 따라서 package_id를 foreign key로 받는다. package_id는 integer type이고 PK이기 때문에 not null이다.

- hazardous_type (PK) : package_id와 함께 primary key로 사용되며 ‘flammable’, ‘explosive’ 와 같이 hazardouse의 내용을 명시한다. 따라서 data type은 VARCHAR(18)로 설정하였고 PK이기 때문에 not null이다.

Shippment

- shippment_id (PK) : shippment의 id를 저장하며, 정수 값이 저장되기 때문에 integer로 설정했다. 여기서 shippment는 transportation, 혹은 warehouse와 같은 특정 운송수단이나 장소를 한 단위로 하며, 하나의 shippment는 여러 package를 가진다. PK이기 때문에 not null로 설정하였다.

Pack_ship

- package_id (PK, FK) : PK이자 FK인 package_id로 integer, not null이다.

- shippment_id (PK, FK) : PK이자 FK인 shippment_id로 integer, not null이다. shippment는 warehouse 혹은 transportation에 다수의 package를 가질 수 있고, package 또한 배송이 완료될 때 까지 다수의 shippment를 거쳐가기 때문에 둘 다 primary key이다.

Warehouse

- warehouse_id (PK) : package가 거쳐가는 warehouse(창고)의 id를 저장한다. 마찬가지로 integer type이며, PK이기 때문에 not null이다. 예를 들어 창고 '301' 과 같이 id가 저장된다.

Transportation

- transportation_id (PK) : package가 거쳐가는 transportation(운송수단)의 id를 저장한다. integer type이며, PK이기 때문에 not null이다.

- type_of_transportation : transportation의 type을 'truck', 'airplane'과 같이 저장한다. 따라서 VARCHAR(18)로 저장하고, not null로 설정하였다.

Ware_visit

- shippment_id (PK, FK) : shippment_id를 저장하며 integer type이고 PK이기 때문에 not null이다.

- warehouse_id (PK, FK) : warehouse_id를 저장하며 integer type이고 PK이기 때문에 not null이다. Ware_visit은 shippment에 해당하는 warehouse의 정보를 저장하며, shippment에 해당하는 warehouse에서 shippment에 포함된 package가 머문 시간을 start_time과 end_time으로 저장해 추후 query에서 package의 위치를 tracking 할 수 있도록하였다.

- start_time : shippment가 warehouse에서 시작된 시간을 저장하며, datetime type으로 저장하였다. 시작시간이기 때문에 not null로 설정하였다. '2023-06-03 09:00:00'와 같이 정보가 들어온다.

- end_time : shippment가 warehouse에서 종료된 시간이고, datetime type으로 저장하였다. 아직 warehouse에 보관 중인 경우 shippment가 종료되지 않았기 때문에 null값을 가진다. 따라서 null을 허용하였다. end_time 또한 '2023-06-03 09:00:00'와 같은 형식으로 정보가 들어온다.

Trans_use

- shippment_id (PK, FK) : shippment_id를 저장하며 integer type이고 PK이기 때문에 not null이다.

- transportation_id (PK, FK) : transportation_id를 저장하며 integer type이고 PK이기 때문에 not null이다. Trans_use은 shippment에 해당하는 transportation의 정보를 저장하며, shippment에 해당하는 transportation에서 shippment에 포함된 package가 transportation에 의해 운반된 시간을 start_time과 end_time으로 저장해 추후 query에서 package의 위치를 tracking 할 수 있도록 하였다.

- start_time : shippment가 transportation으로부터 시작된 시간을 저장하며, datetime type으로 저장하였다. 시작시간이기 때문에 not null로 설정하였다. '2023-06-03 09:00:00'와 같이 정보가 들어온다.

- end_time : shippment가 transportation으로부터 창고 혹은 도착지로 내려진 시간을 저장하며, datetime type으로 저장하였다. 아직 도착하지 않았을 경우 end_time은 null이기 때문에 null을 허용하였다. 마찬가지로 '2023-06-03 09:00:00'와 같이 정보가 들어온다.

Payment

- payment_id (PK) : customer가 pay한 정보를 담는 payment의 id를 저장하므로 integer type으로 설정하였고 PK이기 때문에 not null이다. Payment entity set은 선불여부, 결제 금액, 결제 방식, 결제 날짜, 그리고 payment에 해당하는 customer, shipper, package의 정보를 저장한다.

- is_prepaid : is_prepaid는 선불여부를 boolean type에 저장한다. is_prepaid는 not null로 설정하였다.

- amount : amount는 결제 금액으로, integer type으로 설정하여 '800', '500'과 같이 들어오게 된다. 이 amount의 단위는 달러로 설정하였고, 결제 금액이기 때문에 not null로 설정하였다.

- pay_method : pay_method는 결제 방식으로, VARCHAR(18)로 설정하여 'bill', 'credit card'와 같은 형식으로 들어온다. 결제 방식 또한 not null로 설정하였다.

- pay_date : pay_date는 결제한 날짜를 저장하기 때문에 datetime type으로 설정하였다. pay_date는 not null로 설정하였다.

- customer_id (FK) : pay를 한 customer의 정보를 저장하기 위해 integer type의 customer_id를 foreign key로 받고, not null로 설정하였다.

- shipper_id (FK) : pay를 받은 shipper의 정보를 저장하기 위해 integer type의 shipper_id를 foreign key로 받고, not null로 설정하였다.

- package_id (FK) : pay에 해당하는 package의 정보를 저장하기 위해 integer type의 package_id를 foreign key로 받고, not null로 설정하였다.

Bill

- bill_id (PK) : customer에게 달별로 청구할 bill의 정보를 담는 entity set의 bill id를 저장한다. integer type으로, PK이기 때문에 not null이다.

- bill_date : bill이 customer에게 청구된 날짜를 저장하고, datetime data type, not null로 설정하였따.

- customer_id (FK) : bill이 청구될 customer의 정보를 가지고 있기 위해 integer type의 customer_id를 foreign key로 받고, not null로 설정하였다.

Bill_pay

- payment_id (PK, FK) : bill에 들어갈 integer type의 payment_id를 foreign key로 받고, not null로 설정하였다.

- bill_id (FK) : 여러 payment를 저장할 bill의 정보를 받기 위해 bill_id를 foreign key로 받고 not null로 설정하였다. Bill_pay entity set을 통해 bill에 해당하는 다수의 payment 값을 저장한다.

3. Queries

3-1. CRUD file

```
CREATE TABLE Customer
(
    customer_id     integer  NOT NULL ,
    name            varchar(18)  NOT NULL ,
    address            varchar(50)  NOT NULL ,
    PRIMARY KEY  (customer_id)
);
```

```
CREATE TABLE Shipper
(
    shipper_id      integer  NOT NULL ,
    name            varchar(18)  NOT NULL ,
    account_number  varchar(50)  NOT NULL ,
    PRIMARY KEY  (shipper_id)
);
```

```
CREATE TABLE Service
(
    service_id          integer  NOT NULL ,
    weight              integer  NOT NULL ,
    type_of_package     varchar(18)  NOT NULL ,
    timeliness          datetime  NOT NULL ,
    arrive_time         datetime  NULL ,
    PRIMARY KEY  (service_id)
);
```

```
CREATE TABLE Recipient
(
    recipient_id        integer  NOT NULL ,
    name                varchar(18)  NOT NULL ,
    address             varchar(50)  NOT NULL ,
    PRIMARY KEY (recipient_id)
);
```

```
CREATE TABLE Package
(
    package_id          integer  NOT NULL ,
    is_delivered        boolean  NOT NULL ,
    customer_id         integer  NOT NULL ,
    shipper_id          integer  NOT NULL ,
    service_id          integer  NOT NULL ,
    recipient_id        integer  NOT NULL ,
    PRIMARY KEY (package_id),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY (shipper_id) REFERENCES Shipper(shipper_id),
    FOREIGN KEY (service_id) REFERENCES Service(service_id),
    FOREIGN KEY (recipient_id) REFERENCES Recipient(recipient_id)
);
```

```
CREATE TABLE Shippment
(
    shippment_id    integer  NOT NULL ,
    PRIMARY KEY  (shippment_id)
);
```

```
CREATE TABLE Pack_ship
(
    package_id      integer  NOT NULL ,
    shippment_id    integer  NOT NULL ,
    PRIMARY KEY  (package_id,shippment_id),
    FOREIGN KEY  (package_id) REFERENCES Package(package_id),
    FOREIGN KEY (shippment_id) REFERENCES Shippment(shippment_id)
);
```

```
CREATE TABLE International
(
    customs_declaration_id integer  NOT NULL ,
    package_id         integer  NOT NULL ,
    content            varchar(18)  NOT NULL ,
    value              integer  NOT NULL
        constraint value check (value >= 800),
    PRIMARY KEY  (customs_declaration_id),
    FOREIGN KEY (package_id) REFERENCES Package(package_id)
);
```

```
CREATE TABLE Hazardous
(
    package_id         integer  NOT NULL ,
    hazardous_type     varchar(18)  NOT NULL ,
    PRIMARY KEY  (package_id, hazardous_type),
     FOREIGN KEY (package_id) REFERENCES Package(package_id)
);
```

```
CREATE TABLE Transportation
(
    transportation_id   integer  NOT NULL ,
    type_of_transport   varchar(18)  NOT NULL ,
    PRIMARY KEY  (transportation_id)
);
```

```
CREATE TABLE Warehouse
(
    warehouse_id    integer  NOT NULL ,
    PRIMARY KEY  (warehouse_id)
);
```

```
CREATE TABLE Trans_use
(
    shippment_id        integer  NOT NULL ,
    transportation_id   integer  NOT NULL ,
    start_time          datetime  NOT NULL ,
    end_time            datetime  NULL ,
    PRIMARY KEY  (shippment_id, transportation_id),
    FOREIGN KEY (shippment_id) REFERENCES Shippment(shippment_id),
    FOREIGN KEY (transportation_id) REFERENCES Transportation(transportation_id)
);
```

```
CREATE TABLE Ware_visit
(
    shippment_id        integer  NOT NULL ,
    warehouse_id        integer  NOT NULL ,
    start_time          datetime  NOT NULL ,
    end_time            datetime  NULL ,
    PRIMARY KEY  (shippment_id, warehouse_id),
    FOREIGN KEY (shippment_id) REFERENCES Shippment(shippment_id),
    FOREIGN KEY (warehouse_id) REFERENCES Warehouse(warehouse_id)
);
```

```
CREATE TABLE Payment
(
    payment_id          integer  NOT NULL ,
    is_prepaid          boolean  NOT NULL ,
    amount              integer  NOT NULL ,
    pay_method          varchar(18)  NOT NULL ,
    pay_date            datetime  NOT NULL ,
    customer_id         integer  NOT NULL ,
    shipper_id          integer  NOT NULL ,
    package_id          integer  NOT NULL ,
    PRIMARY KEY (payment_id),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY (shipper_id) REFERENCES Shipper(shipper_id),
    FOREIGN KEY (package_id) REFERENCES Package(package_id)
);
```

```
CREATE TABLE Bill
(
    bill_id         integer  NOT NULL ,
    bill_date       datetime  NOT NULL ,
    customer_id     integer  NOT NULL ,
    PRIMARY KEY  (bill_id),
     FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);
```

```
CREATE TABLE Bill_pay
(
    payment_id      integer  NOT NULL ,
    bill_id         integer  NOT NULL ,
    PRIMARY KEY  (payment_id),
     FOREIGN KEY (payment_id) REFERENCES Payment(payment_id),
     FOREIGN KEY (bill_id) REFERENCES Bill(bill_id)
);
```

먼저 앞서 BCNF decomposition을 진행한 후 만든 physical schema diagram을 기반으로 다음과 같이 table에 대한 create문을 작성하였다. 특별히 추가할 constraint는 없는 걸로 판단하였지만, International table의 경우, 세관신고는 특정 금액 이상의 물품만 세관신고를 하는 것으로 판단하여 value가 800 이상인지 check하도록 구현하였다.

```
insert into Customer values(1001, 'John Smith','123 Main Street, CityA, StateX');
insert into Customer values(1002, 'Jane Doe','789 Oak Drive, CityC, StateZ');
insert into Customer values(1003, 'Emily Johnson','456 Elm Avenue, CityB, StateY');
insert into Customer values(1004, 'Michael Brown','321 Maple Court, CityE, StateW');
insert into Customer values(1005, 'Sophia Lee','987 Pine Lane, CityD, StateV');
insert into Customer values(1006, 'Daniel Johnson','654 Cedar Road, CityF, StateU');

insert into Shipper values(2001, 'Tony Stark', 'SH123456');
insert into Shipper values(2002, 'Won Ik', 'SH789012');
insert into Shipper values(2003, 'Harry Styles', 'SH789012');
insert into Shipper values(2004, 'Emma Watson', 'KM345678');
insert into Shipper values(2005, 'Park Ji-sung', 'KM901234');
insert into Shipper values(2006, 'Alexandra Kim', 'WR567890');
insert into Shipper values(2007, 'frank ocean', 'WR123789');

insert into Service values(3001, 10, 'envelope', '2023-06-02 10:00:00', NULL);
insert into Service values(3002, 5, 'small box', '2023-06-02 14:30:00', '2023-06-03 14:00:00');
insert into Service values(3003, 15, 'medium box', '2023-06-05 08:00:00', NULL);
insert into Service values(3004, 8, 'envelope', '2023-06-04 09:00:00', NULL);
insert into Service values(3005, 12, 'medium box', '2023-06-06 13:30:00', '2023-06-03 14:00:00');
insert into Service values(3006, 20, 'large box', '2023-06-06 15:00:00', NULL);
insert into Service values(3007, 7, 'envelope', '2023-06-07 12:30:00', '2023-06-03 14:00:00');
insert into Service values(3008, 14, 'large box', '2023-06-07 13:00:00', NULL);

insert into Recipient values(4001, 'David Johnson', '789 Maple Avenue');
insert into Recipient values(4002, 'Sarah Williams', '321 Oak Street');
```

```
delete from Shippment;
delete from Package;
delete from Recipient;
delete from Service;
delete from Shipper;
delete from Customer;

drop table Bill_pay;
drop table Bill;
drop table Payment;
drop table Ware_visit;
drop table Trans_use;
drop table Warehouse;
drop table Transportation;
drop table Hazardous;
drop table International;
drop table Pack_ship;
drop table Shippment;
drop table Package;
drop table Recipient;
drop table Service;
drop table Shipper;
drop table Customer;
```

이후 insert, delete, drop까지 구현하였다.

create, insert 문이 들어간 text file은 20191617_1.txt , delete, drop 문이 들어간 text file은 20191617_2.txt 에 저장하여 conn_test.cpp에서 text file을 읽어 sql문을 실행시키도록 구현하였다.

```
CREATE TABLE Customer ( customer_id integer  NOT NULL , name varchar(18)  NOT NULL , addr
CREATE TABLE Shipper ( shipper_id integer  NOT NULL , name varchar(18)  NOT NULL , accour
CREATE TABLE Service ( service_id integer  NOT NULL , weight integer  NOT NULL , type_of_
CREATE TABLE Recipient ( recipient_id integer  NOT NULL , name varchar(18)  NOT NULL , a
CREATE TABLE Package ( package_id integer  NOT NULL , is_delivered boolean  NOT NULL , c
NULL , PRIMARY KEY (package_id), FOREIGN KEY (customer_id) REFERENCES Customer(customer_
FOREIGN KEY (recipient_id) REFERENCES Recipient(recipient_id) );
CREATE TABLE Shippment ( shippment_id integer NOT NULL , PRIMARY KEY  (shippment_id) );
CREATE TABLE Pack_ship( package_id integer  NOT NULL , shippment_id integer  NOT NULL ,
(shippment_id) REFERENCES Shippment(shippment_id) );
CREATE TABLE International ( customs_declaration_id integer  NOT NULL , package_id integ
(customs_declaration_id), FOREIGN KEY (package_id) REFERENCES Package(package_id) );
CREATE TABLE Hazardous( package_id integer  NOT NULL , hazardous_type varchar(18)  NOT N
CREATE TABLE Transportation ( transportation_id integer  NOT NULL , type_of_transport
CREATE TABLE Warehouse( warehouse_id integer  NOT NULL , PRIMARY KEY  (warehouse_id) );
CREATE TABLE Trans_use( shippment_id integer  NOT NULL , transportation_id  integer  NOT
FOREIGN KEY (shippment_id) REFERENCES Shippment(shippment_id), FOREIGN KEY (transportati
CREATE TABLE Ware_visit( shippment_id integer  NOT NULL , warehouse_id integer  NOT NULL
(shippment_id) REFERENCES Shippment(shippment_id), FOREIGN KEY (warehouse_id) REFERENCES
CREATE TABLE Payment ( payment_id  integer  NOT NULL , is_prepaid  boolean  NOT NULL , a
NULL , shipper_id integer  NOT NULL , package_id integer  NOT NULL , PRIMARY KEY (paymen
Shipper(shipper_id), FOREIGN KEY (package_id) REFERENCES Package(package_id) );
CREATE TABLE Bill( bill_id integer  NOT NULL , bill_date datetime  NOT NULL , customer_i
CREATE TABLE Bill_pay( payment_id integer  NOT NULL , bill_id integer  NOT NULL , PRIMAR
Bill(bill_id) );
insert into Customer values(1001, 'John Smith','123 Main Street, CityA, StateX');
insert into Customer values(1002, 'Jane Doe','789 Oak Drive, CityC, StateZ');
insert into Customer values(1003, 'Emily Johnson','456 Elm Avenue, CityB, StateY');
insert into Customer values(1004, 'Michael Brown','321 Maple Court, CityE, StateW');
insert into Customer values(1005, 'Sophia Lee','987 Pine Lane, CityD, StateV');
```

```
delete from Service;
delete from Shipper;
delete from Customer;
drop table Bill_pay;
drop table Bill;
drop table Payment;
drop table Ware_visit;
drop table Trans_use;
drop table Warehouse;
drop table Transportation;
drop table Hazardous;
drop table International;
drop table Pack_ship;
drop table Shippment;
drop table Package;
drop table Recipient;
drop table Service;
drop table Shipper;
drop table Customer;
```

(20191617_1.txt, 20191617_2.txt 캡쳐)

3-2. Query

*** Type I [User input : truck number]**

(Type I-1) Find all customers who had a package on the truck at the time of the crash.

```
/*query 1-1*/
SELECT c.customer_id, c.name
FROM Package p
INNER JOIN Customer c ON p.customer_id = c.customer_id
INNER JOIN Recipient r ON p.recipient_id = r.recipient_id
INNER JOIN Pack_ship ps ON p.package_id = ps.package_id
INNER JOIN Shippment s ON ps.shippment_id = s.shippment_id
INNER JOIN Trans_use tu ON s.shippment_id = tu.shippment_id
INNER JOIN Transportation t ON tu.transportation_id = t.transportation_id
WHERE t.transportation_id = 203
  AND tu.start_time = (
    SELECT MAX(start_time)
    FROM Trans_use
    WHERE transportation_id = 203
  );
```

위의 query문의 경우 c 코드에서는 203 대신 입력 받은 truck number를 target으로 실행하도록 구현하였다. type I-1의 query문의 경우 Package에 Customer, Recipient, Pack_ship, Shippment, Trans_use, Transportation 까지 inner join을 하고, sub query를 통해 transportation_id가 203인 트럭의 shippment 기록 중, start_time이 가장 큰, 즉 가장 최근인 트럭을 찾아서 해당하는 shippment에 package를 가지고 있는 customer의 id와 name을 모두 select 하도록 하였다.

여기서 truck이 사고가 난 시점은 이미 shippment가 종료되어 end_time이 있는 경우에 crash가 날 수 없다고 판단하여 가장 최근에 shippment를 진행하고 있고 아직 shippment를 완료하지 않아 end_time이 null인 truck에 한해서만 crash가 발생한다고 가정하였다. 따라서 위와 같은 query문을 작성하였다.

* 실행 결과

```
Which Type? : 1

---- TYPE I ----

Which truck number? : 203

---- Subtypes in TYPE I ----

        1. TYPE I-1.
        2. TYPE I-2.
        3. TYPE I-3.

Which Sub Type? : 1

** Find all customers who had a package on the truck at the time of the crash **
crashed truck number : 203
[ID] : 1001 [Name] : John Smith
[ID] : 1002 [Name] : Jane Doe
[ID] : 1003 [Name] : Emily Johnson
```

출력은 package를 가지고 있는 customer의 ID와 name을 출력하도록 하였고, 알맞은 결과가 출력되는 것을 확인할 수 있다.

(Type I-2) Find all recipients who had a package on that truck at the time of the crash.

```
/*query 1-2*/
SELECT r.recipient_id, r.name
FROM Package p
INNER JOIN Customer c ON p.customer_id = c.customer_id
INNER JOIN Recipient r ON p.recipient_id = r.recipient_id
INNER JOIN Pack_ship ps ON p.package_id = ps.package_id
INNER JOIN Shippment s ON ps.shippment_id = s.shippment_id
INNER JOIN Trans_use tu ON s.shippment_id = tu.shippment_id
INNER JOIN Transportation t ON tu.transportation_id = t.transportation_id
WHERE t.transportation_id = 203
  AND tu.start_time = (
    SELECT MAX(start_time)
    FROM Trans_use
    WHERE transportation_id = 203
  );
```

마찬가지로 위의 query문의 경우 c 코드에서는 203 대신 입력 받은 truck number를 target으로 실행하도록 구현하였다. query문의 inner join과 sub query의 내용은 type I-1과 동일하고 select하는 부분에서 crash가 발생한 truck에 package를 가지고 있는 recipient의 id와 name을 모두 출력하도록 구현하였다. 사고가 난 상황에 대한 가정도 type I-1과 동일하다.

*실행 결과

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

Which truck number? : 203

---- Subtypes in TYPE I ----

        1. TYPE I-1.
        2. TYPE I-2.
        3. TYPE I-3.

Which Sub Type? : 2

** Find all recipients who had a package on that truck at the time of the crash **
crashed truck number : 203
[ID] : 4001 [Name] : David Johnson
[ID] : 4003 [Name] : kyuho Lee
[ID] : 4004 [Name] : Alexis Brown
```

recipient의 경우에도 ID와 name을 출력하도록 하였고, 알맞은 결과가 출력되는 것을 확인할 수 있다.

(Type I-3) Find the last successful delivery by that truck prior to the crash.

```
/*query 1-3*/
SELECT tu.shippment_id, tu.transportation_id, tu.start_time, tu.end_time
FROM Trans_use tu
WHERE tu.transportation_id = 203
  AND tu.start_time < (
    SELECT MAX(start_time)
    FROM Trans_use
    WHERE transportation_id = 203
  )
  ORDER BY tu.start_time DESC
  LIMIT 1;
```

위의 query문도 c 코드에서는 203 대신 입력 받은 truck number를 target으로 실행하도록 구현하였다. type I-3의 경우 Trans_use table에서 사고가 난 truck은 start_time이 max인, 가장 최근에 운행된 truck 이기 때문에 sub query를 통해 가장 최근 start_time을 찾고, 해당 start_time보다 start_time이 작은 shippment_id와 transportation_id, 그리고 start_time과 end_time을 select하고 start_time을 기준으로 내림차순으로 정렬한 뒤, 가장 상위에 있는 하나의 결과를 뽑는 방식으로 crash가 발생한 truck의 이전 successful delivery 정보를 찾도록 하였다.

*실행 결과

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

Which truck number? : 203

---- Subtypes in TYPE I ----

        1. TYPE I-1.
        2. TYPE I-2.
        3. TYPE I-3.

Which Sub Type? : 3

** Find the last successful delivery by that truck prior to the crash.**
[Shippment_ID] : 102
[Truck] : 203
[Start] : 2023-06-02 11:00:00
[End] : 2023-06-02 16:00:00
```

실행 결과는 이전 successful delivery의 Shippment_id, truck number, start time, end time의 순서로 출력하도록 하였다. 알맞은 출력 결과가 나온 것을 확인할 수 있다.

*** Type II [User input : year]**

(Type II) Find the customer who has shipped the most packages in the past year.

```
/*query 2*/
SELECT c.customer_id, c.name, p.total_payments
FROM Customer c
JOIN (
    SELECT customer_id, COUNT(*) AS total_payments
    FROM Payment
    WHERE YEAR(pay_date) = 2023
    GROUP BY customer_id
) p ON c.customer_id = p.customer_id
WHERE p.total_payments = (
    SELECT MAX(total_payments)
    FROM (
        SELECT COUNT(*) AS total_payments
        FROM Payment
        WHERE YEAR(pay_date) = 2023
        GROUP BY customer_id
    ) temp
);
```

위의 query문 또한 c 코드에서는 2023 대신 입력 받은 year를 target으로 실행하도록 구현하였다. GROUP BY를 통해 Payment 중에서 pay_date가 2023인 경우에 대하여 customer_id와 해당 customer의 payment 횟수를 p.total_payments로 select하고, 이를 Customer table과 customer_id를 통해 join 하도록 하였다. sub query를 통해 total_payments의 최댓값을 구하고 p.total_payments 가 total_payments의 최댓값인 customer를 select하도록 하여 해당 query를 해결하였다.

* 실행 결과

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

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

Which Type? : 2

---- TYPE II ----

** Find the customer who has shipped the most packages in certain year**
Wich Year? : 2023

[ID] : 1002, [Name] : Jane Doe, 2 Times
[ID] : 1006, [Name] : Daniel Johnson, 2 Times
```

결과 출력은 customer_id, name, 그리고 total_payments를 출력 하도록 하였다. 알맞은 결과가 출력되고, 가장 많이 package를 배송한 customer가 다수일 경우에는 여러명을 출력하는 것 또한 확

인할 수 있다.

*** Type III [User input : year]**

(Type III) Find the customer who has spent the most money on shipping in the past year.

```
/*query 3*/
SELECT c.customer_id, c.name, SUM(p.amount) AS total_amount
FROM Customer c
JOIN Payment p ON c.customer_id = p.customer_id
WHERE YEAR(p.pay_date) = 2023
GROUP BY c.customer_id, c.name
HAVING SUM(p.amount) = (
    SELECT MAX(total_amount)
    FROM (
        SELECT SUM(amount) AS total_amount
        FROM Payment
        WHERE YEAR(pay_date) = 2023
        GROUP BY customer_id
    ) temp
);
```

위의 query문 또한 c 코드에서는 2023 대신 입력 받은 year를 target으로 실행하도록 구현하였다. nested sub query를 통해 Payment table을 customer_id로 GROUP BY하여 customer 별 amount의 sum을 구하고 그 중 MAX 값을 구하였다. 이 값을 이용해 MAX total_amount 값을 가지는 customer를 GROUP BY를 통해 구하여 해당하는 customer의 ID, name, total_amount를 select하는 query 문을 구현하였다.

*실행 결과

```
Which Type? : 3

---- TYPE III ----

** Find the customer who has spent the most money on shipping in the past year **
Wich Year? : 2023
[ID] : 1006 , [Name] : Daniel Johnson, [Amount] : 1250
```

출력 결과로는 해당 customer의 ID, name, 그리고 total_amount 까지 출력하도록 하였다. 알맞는 출력이 나오는 것 또한 확인할 수 있다.

*** Type IV [User input : None]**

(Type IV) Find the packages that were not delivered within the promised time.

```sql
/*query 4*/
SELECT p.package_id
FROM Service s
JOIN Package p ON s.service_id = p.service_id
WHERE s.timeliness < s.arrive_time;
```

Type IV query는 Service table에서 Package table을 service_id를 통해 join 하여 timeliness(예상 도착 시간)보다 arrive_time(실제 도착 시간)이 더 큰 package를 모두 select하도록 구현하였다.

*실행 결과

```
Which Type? : 4

---- TYPE IV ----

** Find the packages that were not delivered within the promised time. **
[ID] : 2
```

실행 결과로는 package의 ID를 출력하도록 하였고, 알맞는 결과가 출력되는 것을 확인할 수 있다.

*** Type V [User input : year, month]**

(Type V) Generate the bill for each customer for the past month. Consider creating several types of bills.

각 bill type에 해당하는 query를 설명하기 전 아래의 sql문에서 2023, 6에 해당하는 부분은 c 코드에선 직접 입력받아 해당하는 년월의 bill을 출력하도록 구현하였다.

* Simple bill

```sql
/*simple bill*/
SELECT b.customer_id, c.name, c.address, SUM(p.amount) AS total_amount
FROM Bill b
JOIN Customer c ON b.customer_id = c.customer_id
JOIN Bill_pay bp ON b.bill_id = bp.bill_id
JOIN Payment p ON bp.payment_id = p.payment_id
WHERE YEAR(b.bill_date) = 2023 AND MONTH(b.bill_date) = 6
GROUP BY b.customer_id, c.address;
```

먼저 simple bill의 경우, customer가 지불해야하는 총 amount를 bill에 출력하도록 해야한다. 따라서 Bill table에 Customer, Bill_pay, Payment를 join하여 customer에게 할당된 bill, bill에 포함된 payment를 식별할

수 있도록 하였고, bill의 customer_id와 customer의 address로 GROUP BY하여 customer가 지불해야하는 총 amount를 select할 수 있도록 하였다.

* Service bill

```
/*Service bill*/
SELECT b.customer_id, c.name AS customer_name, bp.payment_id, p.amount, pk.service_id, s.type_of_package
FROM Bill b
JOIN Customer c ON b.customer_id = c.customer_id
JOIN Bill_pay bp ON b.bill_id = bp.bill_id
JOIN Payment p ON bp.payment_id = p.payment_id
JOIN Package pk ON p.package_id = pk.package_id
JOIN Service s ON pk.service_id = s.service_id
WHERE YEAR(b.bill_date) = 2023 AND MONTH(b.bill_date) = 6;
```

다음으로 Service bill의 경우 Service type에 따른 pay 기록을 전부 출력하는 bill이다. 따라서 Customer, Bill_pay, Payment, Package, Service를 join하여 customer의 ID, name, payment ID, packge에 해당하는 amount, service ID와 package의 type 까지 select 하도록 하였다.

* Itemize bill

```
/*Itemize bill*/
SELECT b.customer_id, c.name AS customer_name, p.amount, pk.package_id, s.weight, s.type_of_package
FROM Bill b
JOIN Customer c ON b.customer_id = c.customer_id
JOIN Bill_pay bp ON b.bill_id = bp.bill_id
JOIN Payment p ON bp.payment_id = p.payment_id
JOIN Package pk ON p.package_id = pk.package_id
JOIN Service s ON pk.service_id = s.service_id
WHERE YEAR(b.bill_date) = 2023 AND MONTH(b.bill_date) = 6;
```

마지막으로 Itemize bill에 대하선 customer가 package별로 package의 service정보와 amount를 따로 알 수 있도록 하는 bill이다. 따라서 Service bill과 동일하게 join하여 customer의 ID, name 그리고 amount와 package의 weight, type을 select 하도록 하였다.

type V의 경우 5를 누르고 year, month를 입력하면 세가지 type의 bill을 전부 출력하도록 하였다.

*실행 결과

```
Which Type? : 5

---- TYPE V ----

** Generate the bill for each customer for the past month. Consider creating several types of bills. **
Wich Year? : 2023
Wich Month? : 06
```

```
*** Simple Bill ***
------------------------------------------------------------------
|                                                       2023-06 |
|    ID |           Name |                         Address |  Amout |
|  1001 |      John Smith | 123 Main Street, CityA, StateX |    130 |
------------------------------------------------------------------
------------------------------------------------------------------
|                                                       2023-06 |
|    ID |           Name |                         Address |  Amout |
|  1002 |        Jane Doe |   789 Oak Drive, CityC, StateZ |    320 |
------------------------------------------------------------------
------------------------------------------------------------------
|                                                       2023-06 |
|    ID |           Name |                         Address |  Amout |
|  1003 |   Emily Johnson |  456 Elm Avenue, CityB, StateY |     90 |
------------------------------------------------------------------
------------------------------------------------------------------
|                                                       2023-06 |
|    ID |           Name |                         Address |  Amout |
|  1004 |   Michael Brown | 321 Maple Court, CityE, StateW |    150 |
------------------------------------------------------------------
------------------------------------------------------------------
|                                                       2023-06 |
|    ID |           Name |                         Address |  Amout |
|  1006 |  Daniel Johnson |  654 Cedar Road, CityF, StateU |   1250 |
------------------------------------------------------------------
```

```
*** Service Bill ***
|---------|-----------------|---------|---------|-----------------|-----------------|
|2023-06                                                                          |
|       ID|             Name|  Bill_ID|   Amount|       Service_ID|              Box|
|    1001 |      John Smith |    7001 |     130 |             3001 |        envelope |
|---------|-----------------|---------|---------|-----------------|-----------------|
|2023-06                                                                          |
|       ID|             Name|  Bill_ID|   Amount|       Service_ID|              Box|
|    1002 |        Jane Doe |    7002 |     110 |             3002 |       small box |
|    1002 |        Jane Doe |    7003 |     210 |             3003 |      medium box |
|---------|-----------------|---------|---------|-----------------|-----------------|
|2023-06                                                                          |
|       ID|             Name|  Bill_ID|   Amount|       Service_ID|              Box|
|    1003 |   Emily Johnson |    7004 |      90 |             3006 |       large box |
|---------|-----------------|---------|---------|-----------------|-----------------|
|2023-06                                                                          |
|       ID|             Name|  Bill_ID|   Amount|       Service_ID|              Box|
|    1004 |   Michael Brown |    7005 |     150 |             3005 |      medium box |
|---------|-----------------|---------|---------|-----------------|-----------------|
|2023-06                                                                          |
|       ID|             Name|  Bill_ID|   Amount|       Service_ID|              Box|
|    1006 |  Daniel Johnson |    7007 |    1000 |             3004 |        envelope |
|    1006 |  Daniel Johnson |    7008 |     250 |             3008 |       large box |
|---------|-----------------|---------|---------|-----------------|-----------------|
```

```
*** Itemize Bill ***
|---------|-----------------|---------|---------|---------|-------------|
|2023-06                                                               |
|       ID|             Name|    Amout|  Pack_ID|   Weight|          Box|
|    1001 |      John Smith |     130 |       1 |      10 |     envelope |
|---------|-----------------|---------|---------|---------|-------------|
|2023-06                                                               |
|       ID|             Name|    Amout|  Pack_ID|   Weight|          Box|
|    1002 |        Jane Doe |     110 |       2 |       5 |    small box |
|    1002 |        Jane Doe |     210 |       3 |      15 |   medium box |
|---------|-----------------|---------|---------|---------|-------------|
|2023-06                                                               |
|       ID|             Name|    Amout|  Pack_ID|   Weight|          Box|
|    1003 |   Emily Johnson |      90 |       4 |      20 |    large box |
|---------|-----------------|---------|---------|---------|-------------|
|2023-06                                                               |
|       ID|             Name|    Amout|  Pack_ID|   Weight|          Box|
|    1004 |   Michael Brown |     150 |       5 |      12 |   medium box |
|---------|-----------------|---------|---------|---------|-------------|
|2023-06                                                               |
|       ID|             Name|    Amout|  Pack_ID|   Weight|          Box|
|    1006 |  Daniel Johnson |    1000 |       7 |       8 |     envelope |
|    1006 |  Daniel Johnson |     250 |       8 |      14 |    large box |
|---------|-----------------|---------|---------|---------|-------------|
```

위와 같이 세가지 종류의 Bill이 알맞은 결과로 출력되는 것을 확인할 수 있다.

Query를 처리하는 결과를 명확하게 출력하기 위해 작은 수의 insert문을 이용하여 실행 결과를 보였다. 이후 insert문을 더 큰 사이즈로 추가하여 해당 프로그램의 실행결과는 보고서와 상이할 수 있다. 추가적으로 입력한 data에 대한 설명은 README에 명시하였다.