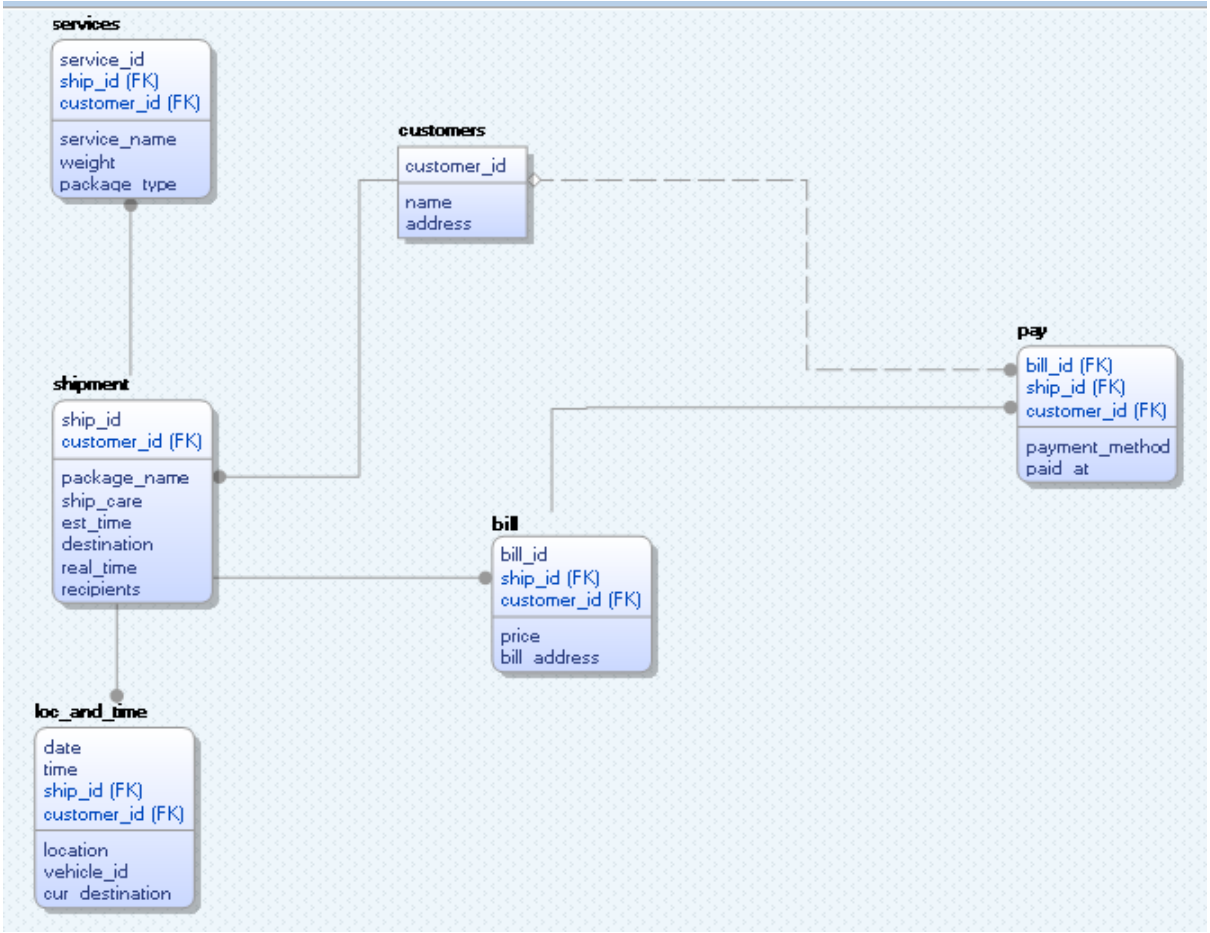
DB_PROJECT2

학과:컴퓨터공학과
학년:3학년
학번:20211524
이름:김준우

1.BCNF



project 1과의 차이점부터 설명하면, shipment부터 알아보자. project 1에서 shipment는 (ship_id, customer_id,package_name,ship_care,est_time,destination)으로 table을 만들어 주었다. 이번에는 query에 맞는 정보를 찾아내기 위해 우선 수령인의 이름을 저장하는 recipient를 새로운 속성으로 만들었고, 예상 도착 날짜인 est_time에 대응되는 실수령일인 real_time이라는 속성을 만들었다. 그리고 그 shipment가 구매가 된 시점을 알려주는 buy_at 이라는 새로운 속성을 만들어 주었다. 따라서 shipment는 (ship_id, customer_id, recipient, package_name, ship_care, est_time, real_time, buy_at, destination)으로 구성된 schema이다.

그 다음 customers는 Project1에서는 bill_id와 ship_id를 FK로 받는 schema였는데, 이 연결을 없애고 customers에는 customer의 필수 정보만 담는 schema로 만들어 주었다. 따라서 customers schema의 속성에서 bill_id와 ship_id를 제거한다. customers schema는 (customer_id, name, address)이다.

세번째로 loc_and_time은 각 제품이 어떤 시점에 어디에 있는지 알아내기 위한 schema이므로 새롭게 ship_id를 FK로 받고, date와 time 그리고 ship_id를 PK로 가지는 schema로 새로 설정해줬다. 따라서 loc_and_time는 (ship_id, date, time,

location, vehicle_id, cur_destination)인 schema다.

schema들의 decomposition을 위해서 BCNF인지 확인해보자. 첫 번째로 shipment는 (ship_id, customer_id, recipient, package_name, ship_care, est_time, real_time, buy_at, destination)로 primary key가 ship_id와 customer_id이다. 여기서 종속성을 찾아보면, id정보가 아닌 속성은 모두 서로 독립적이다. 따라서 BCNF를 해치는 FD가 존재하지 않기 때문에 decomposition을 할 필요가 없다.

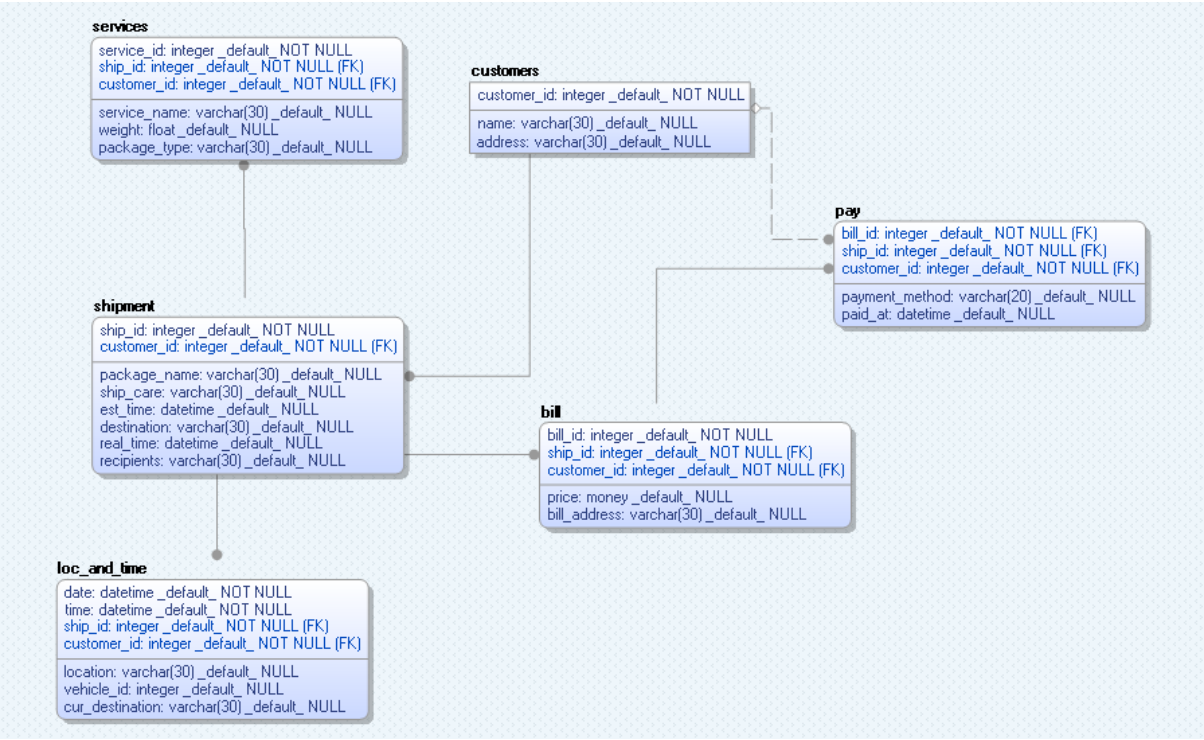
두번째로 customers의 경우에는 (customer_id, name, address)으로 customer_id가 primary key이다. address는 건물 까지만 입력 받도록 설정하여 같은 아파트의 동명이인이 존재할 수 있어서 name과 address에 customer_id가 종속되지 않는다. 따라서 BCNF를 만족해서 decomposition 할 필요가 없다.

세번째로 services는 (service_id, ship_id, customer_id, service_name, weight, package_type) 인데 이 역시 후보키에 포함되지 않는 service_name과 weight 그리고 package_type을 알아도 id 값을 하나로 특정할 수 없어서 종속 되어지지 않아서 BCNF를 해치는 FD가 없기 때문에 decomposition 할 필요가 없다.

loc_and_time도 같은 location, cur_destination, vehicle_id를 모두 알아도 ship_id를 알아낼 수 없어서 종속되어지지 않는다. 따라서 BCNF를 해치는 FD가 없어서 decomposition을 할 필요가 없다.

pay또한 후보키가 아닌 paid_at과 paymethod 에 종속되는 다른 속성이 없기 때문에 decomposition이 필요 없다. bill 또한 같은 고객에게 여러 bill이 갈 수 있기 때문에 bill_address와 price에 종속되는 속성이 존재하지 않기 때문에 decomposition을 할 필요가 없다.

2.physical



erwin으로 만들어낸 physical schema다.

customers부터 보면, 우선 name 과 address는 varchar 길이 30으로 만들었다. 이는 이름이나 주소는 보통 string이고, 바꾸는 경우에 대비해 varchar로 만들었고, customer_id는 고객이 한 번 만들면 주어지는 정수 값이므로 integer로 만들어 주었다. 그리고 id는 primary key이므로 NULL을 허용하지 않는다.

shipment를 보면 우선 id들은 모두 정수라서 integer로 받고 있다. pk인 ship_id와 customer_id만 NULL값을 허용하지 않고 있다. package_name은 상품명이고, ship_care는 care가 필요한지 유무이고, destination은 주소다. 따라서 이 세 속성은 모두 문자열로 구성되어 있기 때문에 varchar형이다. est_time과 real_time은 실제 도착 날짜로 date 자료형을 이용했다.

loc_and_time에는 date는 날짜라서 date, time은 현재 시간이므로 time, location은 현 위치라서 즉, address와 같은 형태라서 varchar, vehicle_id는 id이므로 integer, cur_destination은 마지막으로 도착한 경유지 주소라서 varchar로 만들어 주었다.

services를 보면 id들은 모두 integer로 service_name은 이름이라서 varchar로 weight는 gram 단위의 무게를 의미해서 소수점 까지 포함해야 하기 때문에 float으로 package_type은 박스포장 혹은 스티로폼 포장등을 구분하기 위해 있는 속성이므로 어떤 포장 방식인지는 문자열로 저장하기 때문에 varchar 자료형을 이용했다.

pay는 지불 방법인 payment_method와 지불시기인 paid_at이 있어서 각각 vachar와 date로 만들어줬다.

bill은 청구서로 청구서 id인 bill_id와 지불해야할 돈인 price 그리고 bill이 발송될 주소인 bill_address와 두개의 fk로 구성되어 있다. price는 돈이므로 money 자료형을 이용했고 bill_id는 어느 id가 그렇듯 integer 자료형을 마지막으로 address이므로 varchar를 이용해 만들어 주었다.

mysql에 구현한 table을 보자

```
create table customers
    (customer_id          int,
       name                  varchar(30),
       address               varchar(30),
     primary key (customer_id)
    );

create table shipment
    (ship_id          int,
       customer_id     int,
       recipients          varchar(30),
     package_name     char(30),
     ship_care          varchar(30),
       est_time          date,
       real_time          date,
     buy_at               date,
       destination     varchar(30),
     primary key (ship_id,customer_id),
       foreign key (customer_id) references customers(customer_id) on delete cascade
    );

create table bill
    (bill_id          int,
     ship_id          int,
     customer_id     int,
       price               int,
       bill_address     varchar(30),
     primary key (bill_id, ship_id),
     foreign key (ship_id) references shipment (ship_id)
        on delete cascade,
    foreign key (customer_id) references customers (customer_id) on delete cascade
    );

create table pay
    (bill_id          int,
     ship_id          int,
       customer_id     int,
     payment_method     varchar(30),
     paid_at          date,
     primary key (bill_id, ship_id),
```

```
    foreign key (bill_id) references bill (bill_id)
      on delete cascade,
    foreign key (ship_id) references shipment (ship_id) on delete cascade,
     foreign key (customer_id) references customers (customer_id) on delete cascade
  );

create table services
  (service_id          int,
    ship_id          int,
    customer_id      int,
    service_name     varchar(30),
    weight           float,
    package_type     varchar(30),
   primary key (service_id, ship_id),
   foreign key (ship_id) references shipment (ship_id)
     on delete cascade,
  foreign key (customer_id) references customers(customer_id) on delete cascade
  );

create table loc_and_time
  (ship_id             int,
    customer_id          int.
   date                 date,
    time                 time,
    location             varchar(30),
    vehicle_id             int,
    cur_destination      varchar(30),
   primary key (ship_id,customer_id,date,time),
    foreign key (ship_id) references shipment (ship_id) on delete cascade
    foreign key (customer_id) references customers (customer_id) on delete cascade
  );
```

foreign key가 포함된 table에서 foreign key는 모두 primary key로 사용되어 있기 때문에 on delete cascade를 해주었다. 앞서 physical schema에서 지정한 자료형을 모두 설정해 만들어 주었다.

3.

쿼리를 살펴보자 I-1번은 "SELECT DISTINCT c.customer_id, c.name FROM customers c JOIN shipment s ON c.customer_id = s.customer_id JOIN loc_and_time lt ON s.ship_id = lt.ship_id WHERE lt.vehicle_id = " + to_string(vehicle_id);로 이루어져 있는데 사고가 난 트럭의 id가 vehicle_id이다. 이때, 구해야 하는 것이 customer_id와 name이므로 SELECT를 해주고 고장난 트럭에 있는 shipment를 주문한 고객을 찾아야해서 loc_and_time , shipment, customer를 join해서 찾아주었다.

I-2번은 쿼리는 query = "SELECT DISTINCT s.recipients FROM shipment s JOIN

loc_and_time lt ON s.ship_id = lt.ship_id WHERE lt.vehicle_id = " + to_string(vehicle_id);로 shipment에 저장된 recipient를 찾아야 한다. 따라서 SELECT에는 recipients를 해주고 loc_and_time에 vehicle_id가 있기 때문에 join 을 해줬다.

I-3번은 query = "SELECT cur_destination FROM loc_and_time WHERE vehicle_id = "+to_string(vehicle_id) + " ORDER BY date DESC, time DESC LIMIT 1";
Loc_and_time에 저장된 cur_destination에는 가장 최근에 성공적으로 배송된 주소가 저장되어 있기 때문에 이 값을 찾아야 한다. 따라서 loc_and_time에서 vehicle_id와 입력받은 vehicle_id가 같은 것을 찾아주면 된다.

2번은 query = "SELECT c.customer_id,c.name FROM customers c JOIN shipment s ON c.customer_id = s.customer_id WHERE YEAR(s.buy_at) = " + to_string(year) + " GROUP BY c.customer_id ORDER BY COUNT(*) DESC LIMIT 1";
Year를 입력받아 해당 년도에 가장 많은 물품을 산 고객의 id와 name을 얻어내야 한다. 따라서 customer에 저장된 customer_id와 shipment 에 저장된 customer_id가 같고 shipment에 저장된 year와 같은 해에 구매된 shipment를 찾아서 비교해줘야 한다. 따라서 year와 buy_at에 저장된 년도가 같은지 비교해주고 GROUP BY로 같은 customer가 구매한 제품 개수를 COUNT해준다. 이 값으로 가장 많이 구매한 고객의 정보를 얻어와 출력해준다.

3번은 가장 많은 돈을 사용한 customer를 찾아야 해서 bill에 저장된 price값과 shipment의 buy_at 값을 사용해서 찾아줘야 한다. 따라서 쿼리를 살펴보면
query = "SELECT c.customer_id, c.name FROM customers c JOIN bill b ON c.customer_id = b.customer_id JOIN shipment s ON b.ship_id = s.ship_id WHERE YEAR(s.buy_at) = " + to_string(year) + " GROUP BY c.customer_id ORDER BY SUM(b.price) DESC LIMIT 1";
로 구성되어 있다. 보이듯이 bill과 shipment를 customer와 join하여 찾아주고 있고 이 역시 customer_id 를 이용해 GROUP으로 관리해주고 있다. Bill의 customer_id와 customers의 customer_id 값이 같은 bill 에서의 price를 SUM을 이용해 모두 더해서 찾아주고 있다.

4번째 쿼리는 query = "SELECT ship_id, package_name FROM shipment WHERE est_time<> real_time";
이미 shipment에 real_time 과 est_time이 있기 때문에 이 두 값이 다른 경우를 찾아주면 된다. 따라서 join 이 필요가 없다.