Database System: Project 2(Normalization and Query Processing)

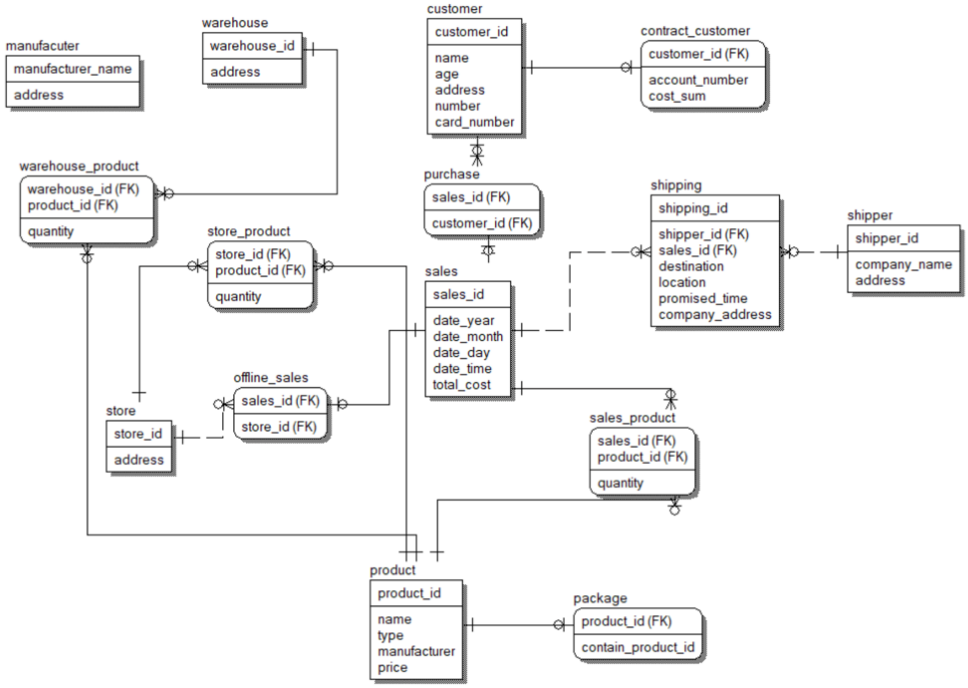
전공: 컴퓨터공학 학년: 3학년 학번: 20201635 이름: 전찬

0. 목차

2. BCNF Decomposition

Project 1을 통해 온라인 / 오프라인 판매에 관련된 DB model을 직접 설계했다. 하지만, 이전에 설계한 model은 normalization이 되지 않은 DB 로, 이 경우에 데이터의 중복이 발생할 수 있으며, DB에서 insert, delete, update 등을 수행할 때에도 문제가 발생할 수 있다. 따라서 DB model을 normalization 하는 것은 중요하며, 이전에 설계한 electronics vendor company의 각 relation schema에 대해 BCNF decomposition을 통해 normalize 된 DB을 구현하는 것이 이번 프로젝트에서 첫 번째로 수행해야 하는 일이다.

우선 Project 1에서 설계한 relation model은 아래와 같다.



위 DB의 각 relation의 이름, 속성, functional dependency, BCNF decomposition, 개선 사항은 아래와 같다.

1. customer : customer_id, name, age, address, number, card_number
customer_id -> name, age, address, number, card_number
customer_id는 customer relation의 primary key이며, super key 이기 때문에, BCNF 형태이다.

2. contract_customer : customer_id, account_number, cost_sum
customer_id -> account_number, cost_sum
이 또한 primary key이기 때문에, BCNF을 만족한다.

3. sales : sales_id, date_year, date_month, date_day, date_time, total_cost
sales_id -> date_year, date_month, date_day, date_time, total_cost
primary key이기 때문에, BCNF을 만족한다.

4. purchase : sales_id, customer_id
sales_id -> customer_id
primary key이기 때문에, BCNF을 만족한다.

5. shipping : shipping_id, shipper_id, sales_id, destination, location, promised_time, company_address
shipping_id -> shipper_id, sales_id, destination, location, promised_time, company_address,
shipper_id -> company_address
으로 shipper_id -> company_address 라는 functional dependency가 BCNF을 만족하지 않는다. 따라서 BCNF decomposition algorithm을 통해 이를 분해할 수 있다.

또한 decomposition에 의해 생성된 위의 shipper relation은 원래 존재하는 shipper relation과 통합할 수 있다. 따라서 shipping에서 company_address을 제거해주면 BCNF을 만족할 수 있다. 따라서 결과는 아래와 같다.

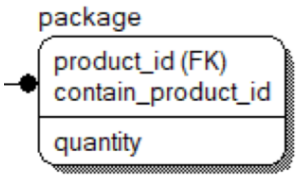
- shipping : shipping_id, shipper_id, sales_id, destination, location, promised_time
- shipper : shipper_id, company_address

6. shipper : shipper_id, company_name, address
shipper_id -> company_name, address
primary key이기 때문에, BCNF을 만족한다.

7. product : product_id, name, type, manufacturer, price
product_id -> name, type, manufacturer, price

primary key이기 때문에, BCNF을 만족한다.

8. package : product_id, contain_product_id
product_id -> contain_product_id
한 product_id = 3인 package(묶음 상품)에 대해, 상품 1, 2가 포함되면 3, 1 / 3, 2 을 tuple로 갖는 relation이 형성될 것이며, product_id가 primary key인 경우 문제가 생길 수 있음을 파악할 수 있다. 따라서 위 relation에서 primary key = contaion_product_id, 그리고 나머지 속성을 product_id, quantity로 설정하는 것이 하나의 개선 사항이 될 수 있다. 여기에서 quantity란 contain_product_id에 해당하는 product가 해당 package에 몇 개 존재하는지를 파악할 수 있는 속성이다. 이와 같이 변경하는 경우 만들어지는 형태는 아래와 같다.



9. sales_product : sales_id, product_id, quantity
sales_id, product_id -> quantity
primary key이기 때문에, BCNF을 만족한다.

10. store : store_id, address
store_id -> address
primary key이기 때문에, BCNF을 만족한다.

11. offline_sales : sales_id, store_id
sales_id ->store_id
primary key이기 때문에, BCNF을 만족한다.

12. warehouse : warehouse_id, address
warehouse_id -> address
primary key이기 때문에, BCNF을 만족한다.

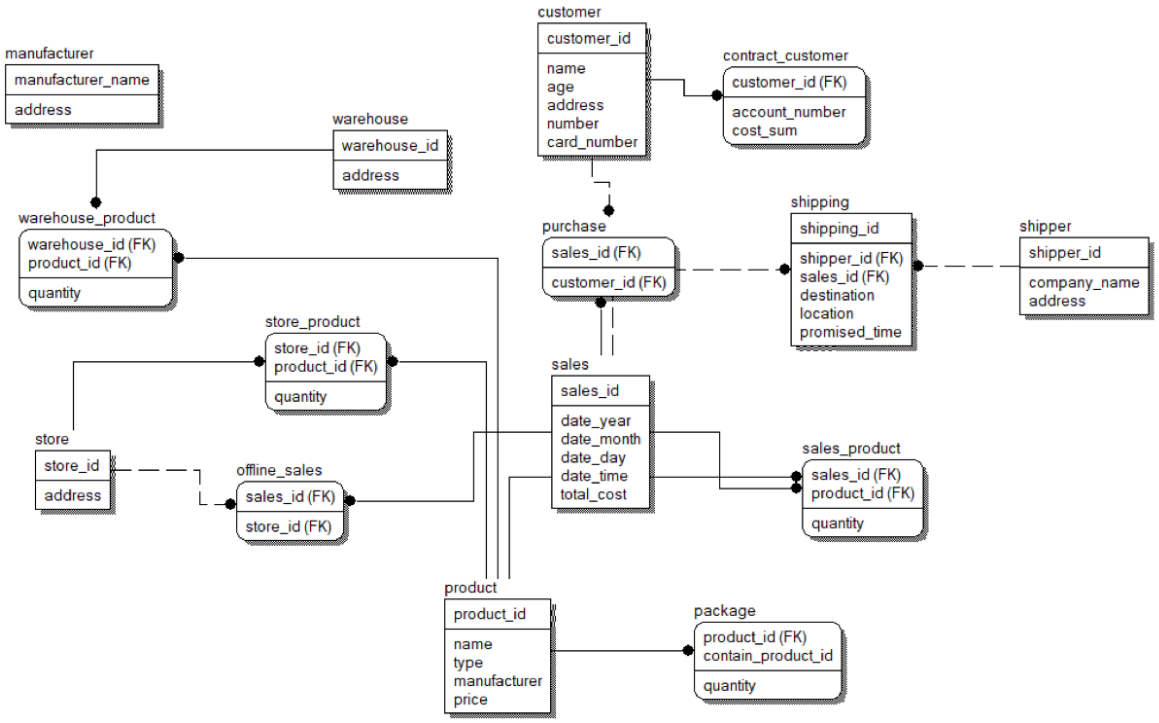
13. manufacturer : manufacturer_name, address
manufacturer_name -> address
primary key이기 때문에, BCNF을 만족한다.

14. store_product : store_id, product_id, quantity
store_id, product_id -> store_id, product_id, qunatity
primary key이기 때문에, BCNF을 만족한다.

15. warehouse_product : warehouse_id, product_id, quantity
warehouse_id, product_id -> warehouse_id, product_id, quantity
primary key이기 때문에, BCNF을 만족한다.

3. Physical Schema Diagram

따라서 위에서 설계한 BCNF을 만족하는 DB model을 ERWIN을 바탕으로 Physical schema diagram 으로 만들어낼 수 있다. 기본적으로 id는 CHAR(5)로, 개수, 나이 등의 count는 INT, 가격 등의 값은 FLOAT, 그리고 주소 등의 문자열은 VARCHAR(18)로 설정하였다. 또한 모든 relation에 대해 NULL / NOT NULL을 설정해 주었으며, 예를 들어, customer에서 필수적이지 않은 name, age 등은 NULL 값을 허용할 수 있도록 설정해주었다. 이를 바탕으로 설계한 DB model은 아래와 같다.



4. Queries

이번 프로젝트에서 마지막으로 할 것은 embedded SQL을 응용해서 직접 instance와 함께 DB를 구현하며, 이에 알맞은 query문을 작성하는 것이다. 우선 첫 번째로, MySQL 문법을 바탕으로 DB을 구현해야 한다. 기본적으로 제공된 코드를 살펴보면, project라는 DB가 이미 존재하며, 이 안에 schema, instance 등을 구현하면 되는 형태인데, 개인적으로 구현한 DB model의 핵심은 customer, sales, product, shipping 이라 할 수 있다. 따라서 이를 위에서 구현한 physical schema diagram을 토대로 txt file에서 CREATE TABLE 을 통해 table을 구현하며, INSERT INTO을 통해 데이터를 삽입하며 실제 DB를 구현할 수 있다. 하지만 개인적인 프로그램 상 오류로 visual studio에서 에러가 발생하였고, 조교님에게 문의하며 해결 방법을 계속해서 찾아보았으나, 해결할 수 없었다. 따라서 실제 DB와 상호작용할 수 없다 보니 직접 DB 구현, DB Query문 구현 등을 수행할 수는 없었다. 또한 query 문 등을 테스트할 수도 없었다. 따라서 위에서 설계한 DB model을 바탕으로 예상되는 Query 형태를 아래 서술하였다.

1.
```
select number
from customer natural join purchase natural join sales natural join shipping
where shipping_id = 'X'
```

1-1.
```
// 데이터가 저장되어 있다고 가정
select product_id
from shipping natural join sales natural join sales_product
where shipping_id = 'X'
INSERT into shipping VALUES (new_ shipping_id, shipper_id, sales_id, destination, location, promised_time)
```

2.
```
select customer_id
from customer join purchase on customer_id join sales on sales_id
group by customer
having sum(total_cost) = max(sum(total_cost))
```

2-1
```
select product_id
from sales_product join purchase
```

```
where customer_id = select customer_id
            from customer join purchase on customer_id join sales on sales_id
group by product_id
having sum(quantity) = max(sum(quantity))
```

3.

```
select product_id
from sales natural join sales_product
where date_year = '2021'
```

3-1

```
select TOP k product_id
from sales_product natural join product
group by product_id
order by sum(quantity)*price
```

3-2

```
select count(product_id)
from product // 총 개수 중 10%을 알아내기 위한 query 문

select TOP i product_id
from sales_product natural join product
group by product_id
order by sum(quantity)*price
```

4.

```
select product_id, sum(quantity)
from sales natural join sales_product
where date_year = '2021'
group by product_id
```

4-1.

```
select TOP k product_id
from sales_product natural join product
```

group by product_id
order by sum(quantity)

4-2.
select count(product_id)
from product // 총 개수 중 10%을 알아내기 위한 query 문

select TOP i product_id
from sales_product natural join product
group by product_id
order by sum(quantity)

5.
select product_id
from store_product
where store_id = '10001' and quantity = 0 and store_id = '10001' and quantity = 0

6.

7.
select customer_id, cost_sum
from contract_customer