Database System: Project 2(Normalization and Query Processing)

전공: 컴퓨터공학 학년: 3학년 학번: 20201635 이름: 전찬

**0. 목차**

1. 구현 목표

2. BCNF Decomposition

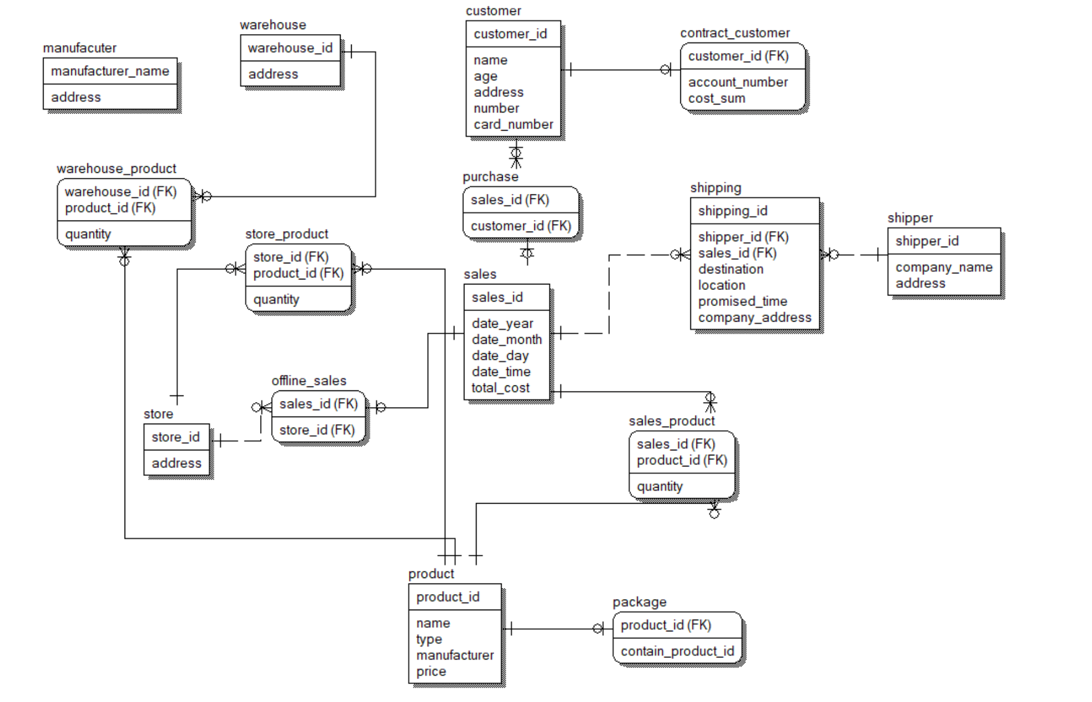
3. Physical Schema Diagram

4. Queries

**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

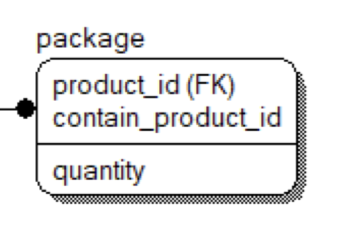
product\_id -> name, type, manufacturer, price

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.

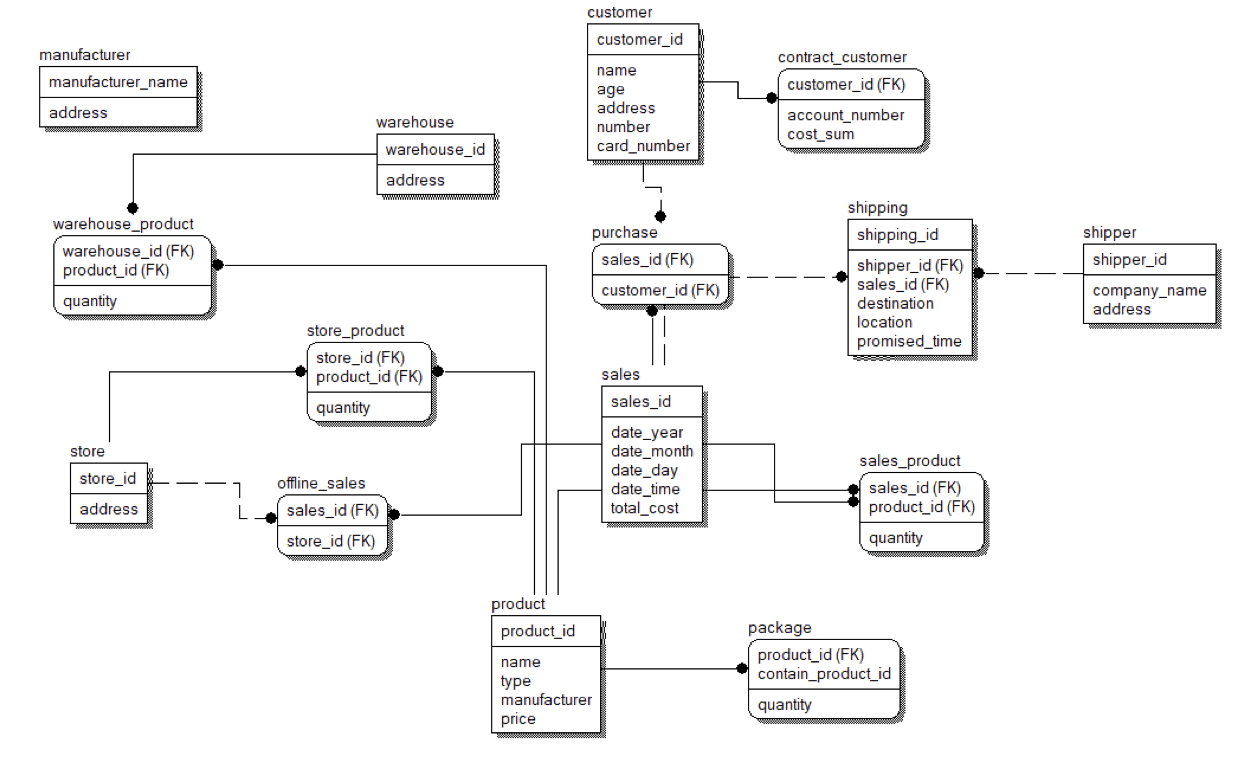
15. warehouse\_product : warehouse\_id, product\_id, quantity

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