Database System

Project2

학번 : 20181662

학과 : 컴퓨터공학과

이름 : 이건영

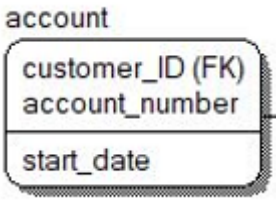
1. BCNF Decomposition

Project 1 의 Schema 가 BCNF 를 만족하는지 확인하고, 그렇지 않은 경우 적절하게 decompose 하여 BCNF 로 만드는 과정이다.

(편의상 attribute 를 순서에 따라 A~Z 로 표시하였다.)

(a->b 가 trivial 한 것을 1 번 조건, a 가 R 의 superkey 인 것을 2 번 조건이라고 부르겠다.)

1) account

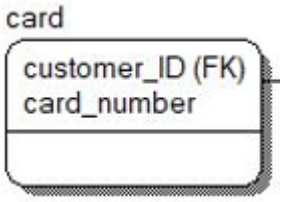


Functional dependency : AB -> C

A 와 B 가 account 의 primary key 이므로 2 번 조건을 만족한다.

따라서 BCNF 를 만족한다.

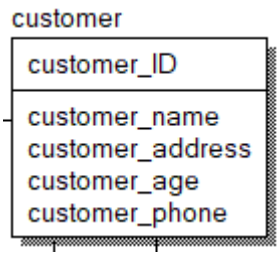
2) card



Functional dependency : None

모든 attribute(A 와 B)가 superkey 이므로 BCNF 를 만족한다.

3) customer

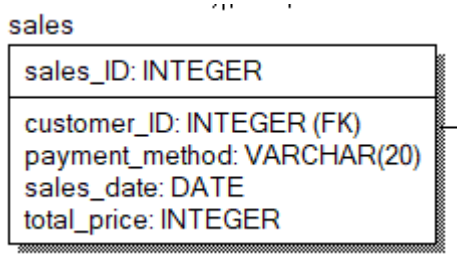


Functional dependency : A -> BCDE

A 가 superkey 이므로 조건 2 를 만족한다.

따라서 BCNF 를 만족한다.

4) sales



Functional dependency : A -> BCDE

A 가 superkey 이므로 조건 2 를 만족한다.

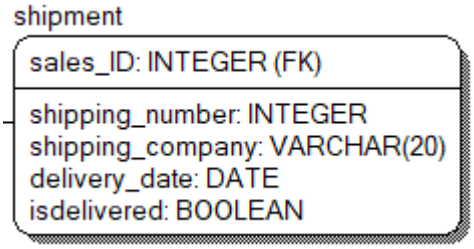
따라서 BCNF 를 만족한다.

5) shipment

Functional dependency : AB -> C

Project 1 에서 배송 정보에 관한 table 을 위와 같이 구성한 이유는 한 구매 내역에 대해 다양한 배송 정보(제품 별로 배송 정보가 다른 경우 등)가 있는 경우를 고려하여 위와 같이 두 개의

primary key 로 구성하였는데, project 2 의 명세를 확인한 후 해당 경우에 대한 고려는 불필요하다고 판단하여 다음과 같이 수정하였다.



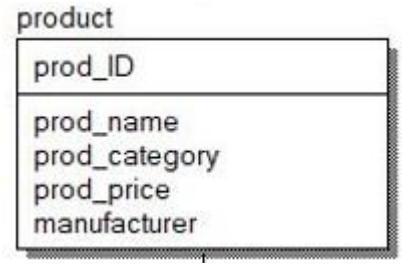
Functional dependency : A -> BC

A 가 superkey 이므로 조건 2 를 만족한다.

따라서 BCNF 를 만족한다.

추가적으로 query 에서 사용하기 위해 배송되었는지 여부를 확인하는 isdelivered 와 shipping_company 라는 속성을 추가하였다.

6) product

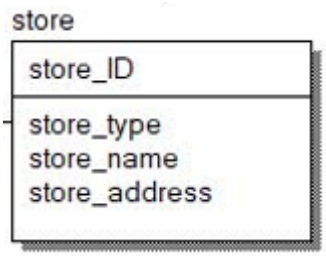


Functional dependency : A -> BCDE

A 가 superkey 이므로 조건 2 를 만족한다.

따라서 BCNF 를 만족한다.

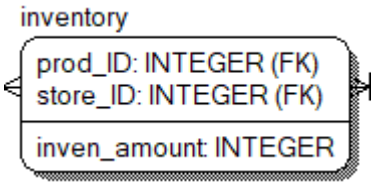
7) store



Functional dependency : A -> BCD

A 가 superkey 이므로 조건 2 를 만족한다. 따라서 BCNF 를 만족한다.

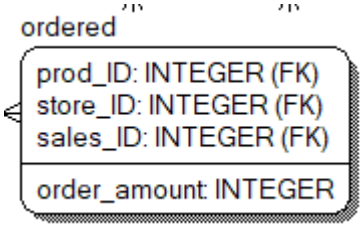
8) inventory



Functional dependency : AB -> C

A, B 가 superkey 이므로 조건 2 를 만족한다. 따라서 BCNF 를 만족한다.

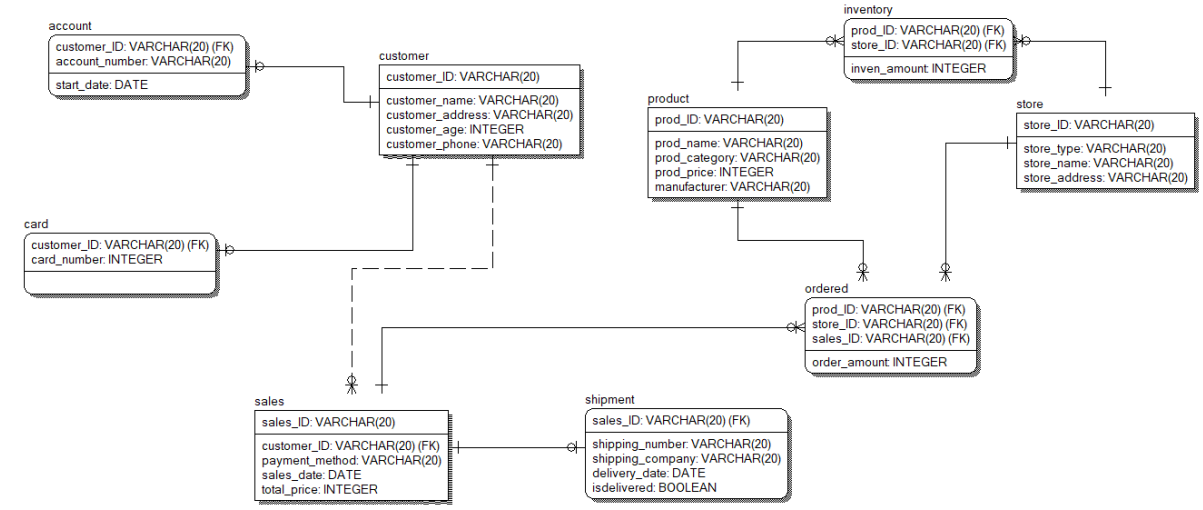
9) ordered



Functional dependency : ABC -> D

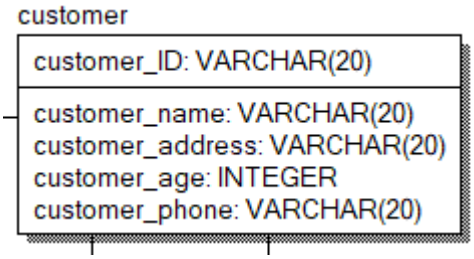
A, B, C 가 superkey 이므로 조건 2 를 만족한다. 따라서 BCNF 를 만족한다.

수정된 후의 Schema Diagram 은 다음과 같다.



2. Description about physical schema diagram

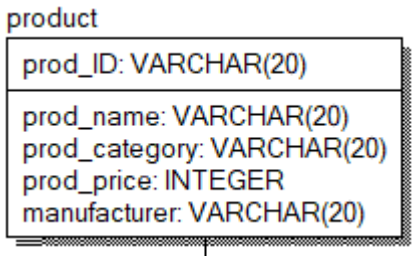
1) customer



고객의 정보를 저장하는 table 이다. 나이인 age 는 INTEGER, 나머지 이름과 주소, 연락처는 문자열 데이터 타입이다. 고객 구분을 위한 ID 와 이름이 있으며 그 중 ID 를 Primary key 로 두었다. 배송지 정보를 위한 주소를 알기 위해 address 를 속성으로 가지고 있다. 또 query type 1 과 같이 특수한 경우 고객에게 연락을 위한 연락처 phone 을 속성으로 추가하였다. 그 외에 나이인 age 가 존재한다.

이름과 ID 는 필수적인 정보로 Not NULL 이며, 나머지 정보는 결제 방식(현장 결제 등)에 따라 없을 수 있는 정보라 판단하여 NULL 이 가능하도록 하였다.

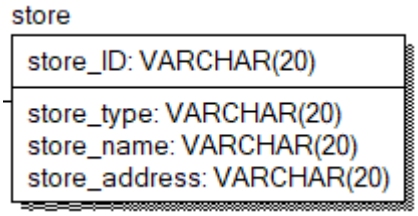
2) product



상품의 정보를 저장하는 table 이다. Primary key 인 ID 가 존재하며, 문자열인 상품명과 category, 제조사가 존재한다. 상품의 가격도 price 로 INTEGER 타입으로 존재한다. 모든 항목이 필수적이라고 판단하여 Not NULL 로 설정했다.

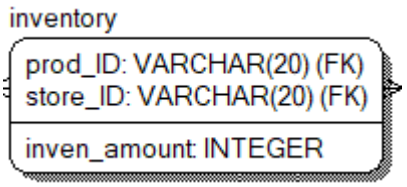
상품의 재고 현황에 관해서는 해당 테이블에서 관리할 경우 매장 별 재고 현황에 대해 관리하기 힘들어진다고 판단하여 별도의 table 을 두었다.

3) store



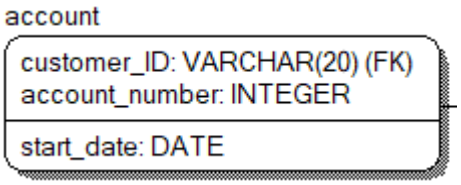
매장의 정보를 저장하는 table 이다. Primary key 로 ID 가 존재하며, 매장의 type, 이름, 주소가 문자열로 저장된다. 매장 type 이란 해당 매장이 On/Offline 인지 구별하기 위해 존재하며 online / offline 둘 중 하나의 값만 가질 수 있는 Constraint 가 존재한다. 모든 속성은 Not NULL 이다.

4) inventory



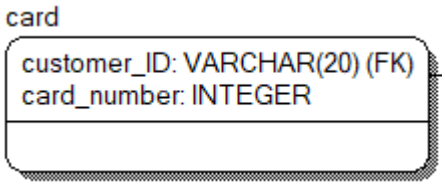
제품의 재고를 매장 별로 관리하기 위해 만든 table 이다. 두 개의 foreign key 인 제품 ID 와 가게 ID 를 primary key 로 가지며, 제품 수량을 INTEGER 타입으로 가진다. 제품 수량은 Not NULL 이며, 최소값이 0 인 Constraint 를 가진다.

5) account



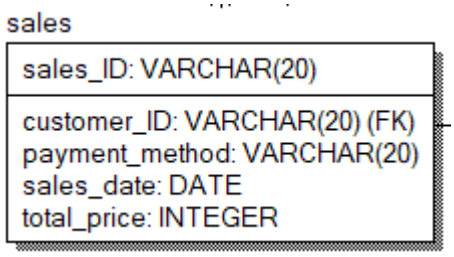
정기적으로 결제하는 고객의 경우 계좌 정보가 필요하다. 이를 저장하기 위한 속성을 customer 에 저장할 경우 정기 결제 고객이 아닌 부분에 대해서 NULL 인 부분이 많아질 것이라 판단하여 따로 table 을 두어 관리하기로 하였다. 계좌 번호와 Foreign key 인 고객 ID 를 primary key 로 가지며, 추가적으로 정기 결제를 시작한 날짜를 담는 DATE 타입의 start_date 속성을 두었다.

6) card



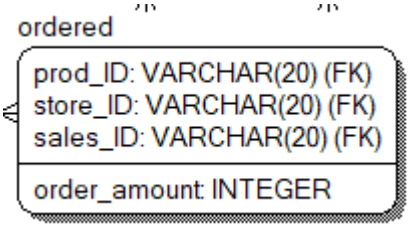
카드로 결제한 고객의 카드 정보를 담는 table 이다. 계좌 정보와 같은 이유로 추가적인 table 을 만들었으며 카드 번호와 고객 ID 를 primary key 로 갖는다.

7) sales



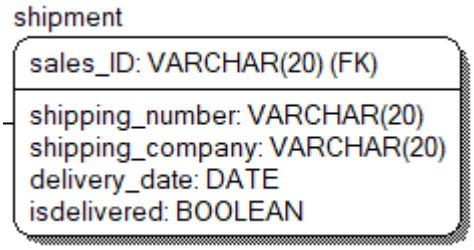
결제가 일어난 정보를 저장하는 table 이다. 따로 sales_ID 를 두어 primary key 로 설정하였다. 결제 방식은 account, card, cash 만을 허용하는 Constraint 가 있다.

8) ordered



결제 내용과 관련하여 어떤 매장에서 어떤 제품이 몇 개 출고되었는지를 저장하는 table 이다. 해당 정보를 다른 sales 등 다른 table 에서 관리할 경우 정보의 중복이 더욱 많아질 것 같아서 이같이 분리하였다. 제품, 가게, 결제 ID 를 foreign key 로 가져와 primary key 로 사용하며, 각 경우의 수량을 Not NULL 인 INTEGER 타입으로 저장한다.

9) shipment



상품의 배송 정보를 담는 table 이다. 현장 결제와 같이 고객이 상품을 직접 수령하는 경우도 있기에 sales table 에 해당 정보를 저장하면 NULL 값이 많이 발생할 것 같아서 분리하였다. 결제 ID 를 primary key 로 가지며, 배송 번호를 INTEGER 데이터타입의 속성으로 가지며 배송 예정일과 배송 회사, 배송이 되었는지 여부를 나타내는 속성을 가진다.

3. Queries and C code implementation

프로젝트 폴더 내에 README 를 작성하였다.

1) CRUD

CRUD 를 위한 txt 파일을 생성과 해체를 담당하는 두 파일로 나눴으며, 구현은 다음과 같다.

```
FILE* fp = fopen("20181662_ci.txt", "r");
fseek(fp, 0, SEEK_END);
size = ftell(fp);
buffer = (char*)malloc(size + 1);
memset(buffer, 0, size + 1);
fseek(fp, 0, SEEK_SET);
fread(buffer, size, 1, fp);
```

위와 같이 파일 전체를 buffer 에 저장한다.

```
char query[2048];
int state = 0;
const char* temp = strtok(buffer, ";");
while (temp != NULL) {
    state = 0;
    state = mysql_query(connection, temp);
    temp = strtok(NULL, ";");
}
```

이후 MySQL 서버에 정상적으로 연결되면 위처럼 ";"를 기준으로 query 들을 구분하여 실행함으로 테이블 생성과 데이터 입력을 수행한다.

```
FILE* fp2 = fopen("20181662_dd.txt", "r");
char* buffer2 = NULL;

fseek(fp2, 0, SEEK_END);
size = ftell(fp2);
buffer2 = (char*)malloc(size + 1);
memset(buffer2, 0, size + 1);
fseek(fp2, 0, SEEK_SET);
fread(buffer2, size, 1, fp2);

temp = strtok(buffer2, ";");
while (temp != NULL) {
    state = 0;
    state = mysql_query(connection, temp);
    temp = strtok(NULL, ";");
}
mysql_close(connection);
```

테이블과 데이터 삭제 시에도 위와 비슷한 과정을 거쳐 query 를 실행시킨 후 MySQL 서버와의 연결을 종료한다.

2) query

(Type 1)

```
select customer_phone, customer_name, customer_ID, customer_address

from shipment natural join sales natural join customer

where shipping_number = X and shipping_company = "USPS";
```

우선 customer 와 shipment 테이블을 이어주는 역할인 sales 테이블을 포함하여 세 테이블을 natural join 하였다. 각각 공통되는 속성이 한 가지이기 때문에 문제없이 join 되었다. 이후 입력받은 배송번호에 해당하는 값과 운송 회사가 USPS 인 행들을 선택하여 출력하였다.

(Type 1-1)

```
select * from shipment where shipping_number = X;
```

배송번호가 X 인 행의 정보를 출력한 뒤, 다음의 query 로 내용을 update 하였다.

```
update shipment

SET delivery_date = delivery_date + 7

where shipping_number = X and shipping_company = "USPS";
```

배송 기한을 일주일 연장시켰다.

(Type 2)

```
select customer_ID, customer_name, sum(total_price)

from sales natural join customer

where year(sales_date) = 2021

group by customer_ID

order by sum(total_price) desc

limit 1;
```

customer 와 sales 테이블을 natural join 한 후, 판매 년도가 2021 년인 행들에 대해서 고객 ID 별 구매 금액을 합산하여 내림차순으로 정렬하였다. 마지막으로 상위 1 줄만을 표시하게 하여 최대값을 구했다.

(Type 2-1)

```
select prod_ID, sum(order_amount) as amount

from sales natural join ordered

where customer_ID = "mostID"

group by prod_ID

order by amount desc

limit 1;
```

방금 전 결과의 customer_ID 를 mostID 변수에 저장한 후, 해당 고객 ID 로 sales 목록을 불러와 제품 별로 구매 개수를 합산한 후 내림차순으로 정렬하였다. 마찬가지로 상위 1 개의 제품명만을 출력하였다.

(Type 3)

```
select prod_ID, prod_price, order_amount, sum((prod_price * order_amount))

from sales natural join ordered natural join product

where year(sales_date) = 2021

group by prod_ID

order by prod_ID asc;
```

sales 와 ordered, product 를 natural join 하였다. 그 후 2021 년의 판매 내역을 대상으로 제품별 판매량과 제품가격을 곱해 총 매출액을 계산하였다. 이를 정렬하여 출력하였다.

이 과정에서 row 를 출력할 때마다 카운트를 하여 총 row 의 개수를 numRow 에 저장하였다.

(Type 3-1)

```
select prod_ID, prod_price, order_amount, sum((prod_price * order_amount)) as k

from sales natural join ordered natural join product

where year(sales_date) = 2021

group by prod_ID

order by k desc

limit K;
```

K 를 사용자로부터 입력받은 후, Type 3 의 query 에서 정렬을 매출액 기준 내림차순으로, limit 를 K 로 설정한 후 출력하였다.

(Type 3-2)

```
select prod_ID, prod_price, order_amount, sum((prod_price * order_amount)) as k

from sales natural join ordered natural join product

where year(sales_date) = 2021

group by prod_ID

order by k desc

limit 10%;
```

Type 3-1 과 비슷하나 limit 부분이 다른데, 방금 계산한 numRow 의 10%를 계산하여 limit 의 값으로 사용하였다.

(Type 4)

```
select prod_ID,   sum(order_amount)
from sales natural join ordered natural join product
where year(sales_date) = 2021
group by prod_ID
order by prod_ID desc;
```

Type 3 과 비슷하나 판매 수량의 합을 제품별로 구해서 출력하였다.

(Type 4-1)

```
select prod_ID,   sum(order_amount) as k
from sales natural join ordered natural join product
where year(sales_date) = 2021
group by prod_ID
order by k desc
limit K;
```

입력받은 K 에 대해 상위 K 개의 행을 판매 수량의 합을 기준으로 정렬하여 출력하였다.

(Type 4-2)

```
select prod_ID,   sum(order_amount) as k
from sales natural join ordered natural join product
where year(sales_date) = 2021
group by prod_ID
order by k desc
limit 10%;
```

Type 3-2 와 비슷하게 Type 4 에서 계산한 numRow 를 이용하여 그 10%에 해당하는 행 까지만 출력하였다.

(Type 5)

```
select prod_ID, inven_amount

from inventory natural join store

where store_address = "California" and inven_amount = 0;
```

가게 별 재고 현황을 파악하기 위해 inventory 와 store 테이블을 natural join 하였다. 이후 주소가 California 에 있으면서 수량이 0 인 제품 ID 를 찾아 출력하였다.

(Type 6)

```
select *

from shipment

where delivery_date < '2022-06-08' and isdelivered = 0;
```

shipment 테이블에 대해서 배송이 완료되지 않았으며, 배송 예정일이 지난 행들을 찾아 출력하였다. SQL 에서 GETDATE() 함수를 사용하여 현재 날짜를 받아올 수 있다는 정보를 찾았으나 MySQL 에서 작동하지 않아 프로젝트 제출 마감 날짜를 기준으로 query 를 구현하였다.

(Type 7)

```
select customer_ID, sum(total_price) as price

from sales

where year(sales_date) = '2022' and month(sales_date) = '05'

group by customer_ID;
```

제출 마감일을 기준으로 한 달 전인 22 년 5 월에 대해서 고객 별 구매 총액을 출력하였다.