[project2]20170024 김병천

1. Logical schema diagram.

이 데이터베이스에 필요한 functional dependency는 다음과 같다.

product_ID->type, price,manufacturer,manufacturing_year
product_ID,component_ID->amount
store_ID->address
store_ID,product_ID->stock_amount
customer_ID->account,card_number,phone_number
payment_ID->store_ID,product_ID,customer_ID,payment_year&month&day
payment_ID,tracking_number->due_year&month&day,delivered_year&month&day

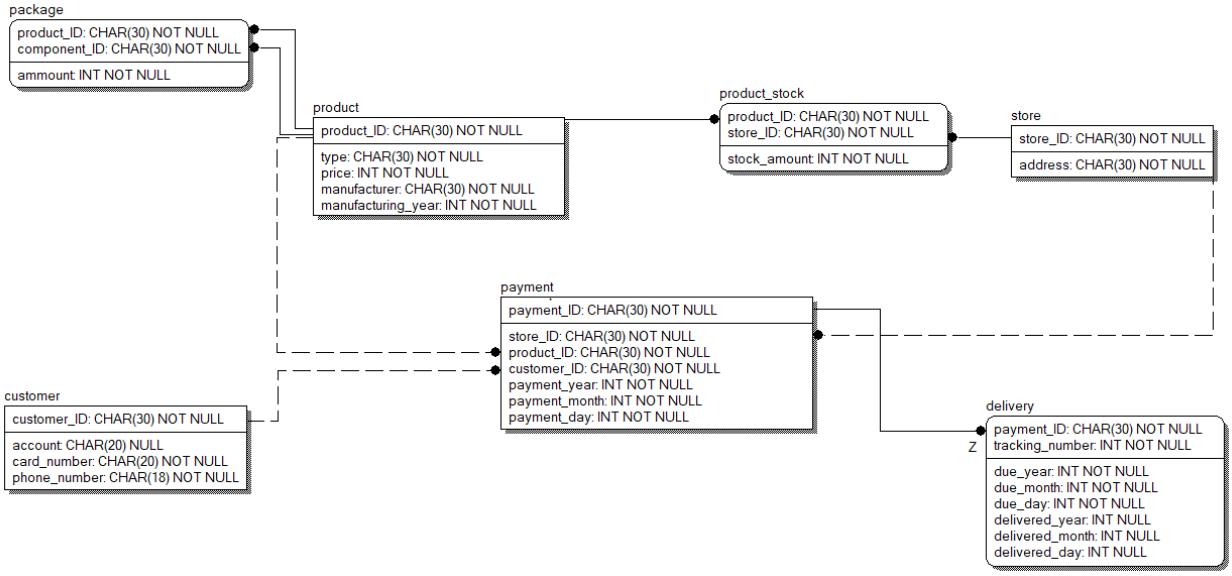
기존에 작성한 논리적 스키마에서 BCNF를 위반한 것이 없기에 분할한 relationship은 없다.
그러나 기존의 logical schema diagram에 있는 정보로는 sql문을 작성할 수 없기에 몇몇 relationship에 몇몇 속성들을 추가했다.
또, 검색의 편의를 위해 몇몇 relationship의 속성을 조금 수정하였다.

다음은 수정한 relationship 및 속성의 목록이다.

1. Package
    1. ammount 추가
        해당 패키지에 구성요소가 몇개가 들어있는지를 알 수 없었기 때문에, 해당 구성요소가 몇개가 들어있는지 확인할 수 있는 속성을 추가했다.
        이로서 휴지 패키지를 휴지 10개의 세트로 구성하는 것이 가능해졌다.
2. Customer
    1. phone_number 추가
        해당 고객의 연락처를 알 수 있는 속성이 기존에 없었기 때문에, 고객의 연락처를 알 수 있는 속성을 추가했다.
3. Store
    1. Address 추가
        점포의 주소를 알 수 있는 속성이 기존에 없었기 때문에, 해당 점포의 주소를 알 수 있는 속성을 추가했다.
4. Payment
    1. price 제거
        payment를 조회할 경우, product와의 join 없이도 그 주문의 가격을 알 수 있게 하고싶었다.
        때문에 product의 price 속성을 payment에 외부속성으로 가져오고 싶었다.
        그러나 price가 product 테이블 내에서 primary key가 아니기 때문에 price 속성을 payment 테이블 내에서 제거했다.
    2. payment_day를 payment_year,payment_month,payment_day로 세분
        배송일정과 관련된 정보를 편리하게 처리하기 위해 주문일자를 세분화했다.
5. delivery
    1. due_date/delivered_date를 각각 4-2처럼 세분화
        배송예정일/실제 배송일과 관련된 정보를 편리하게 처리하기 위해 세분화했다.

2. Physical schema diagram

1. Product
    1. Primary key를 포함한 모든 속성에 대한 정보가 필요하기 때문에 모두 not null로 선언했다.
    2. 가격정보와 제조년도는 정수로, 그 이외의 정보들은 문자열로 표현하도록 했다.
2. Customer
    1. 계좌정보/카드정보/연락처는 모두 문자열로 표기한다.
    2. 계좌가 있을 경우, monthly pay로 결제하고, 계좌가 없을 경우 카드넘버로 결제한다. 결제를 하기위해서는 기본적으로 카드넘버가 필요하기에 카드넘버는 not null로한다. 계좌는 있어도 없어도 상관없기 때문에 null을 허용한다.



3. Phone number는 유사시의 연락을 위해 꼭 필요함으로 not null로한다.

3. Store
   1. 점포가 있다면 그 점포가 위치한 주소가 반드시 있기 때문에 not null로한다.
4. Package
   1. 특정 패키지의 구성요소임은 곧 그 패키지 내에 그 구성요소가 하나 이상은 있음을 의미하니 not null로한다.
   2. 한 제품은 여러 패키지의 구성요소일 수 있기 때문에 product.product_ID - component_ID의 관계는 one to many이다.
5. Product_stock
   1. 한 상품은 여러 점포의 재고로서 존재할 수 있기에 product-product_stock에서 product_ID는 one to many의 관계이다.
   2. 한 점포에 여러 상품의 재고정보가 존재할 수 있기에 store-product_stock에서 store_ID는 one to many의 관계이다.
   3. 해당 점포의 특정 물품의 재고량은 정수이기 떄문에 int로 표기한다.
   4. 또한 해당 점포에 한번도 들어오지 않은 경우는 재고등록을 하지 않으면 되고, 재고가 다 떨어져도 0으로 표기하면 되기 때문에 not null로 표기한다.
6. Payment
   1. 한 사람이 여러개의 제품을 구매하더라도, 한 제품씩 payment로 기록된다. 특정 제품은 안팔릴 수도 있고 여러번 팔릴 수도 있기에 product-payment에서의 product_ID는 one to many의 관계이다.
   2. 한 고객이 제품을 안살수도 있고, 여러개를 살수도 있기 때문에 Customer- payment에서의 customer_ID는 one to many이다.
   3. 한 지점에서 제품이 안팔릴수도 있고 여러개가 팔릴 수도 있기 때문에 store_payment에서의 store_ID는 one to many이다.
   4. 어떤 제품을 구매했다면 반드시 구매한 날짜가 있기 때문에 구매 날짜 관련 항목들은 not null이다.
7. Delivery
   1. payment는 delivery의 identifying relationship이다. 각 payment마다 delivery는 존재할 수 있고, 존재하지 않을 수 있지만, 존재한다면 반드시 단 하나만 존재하기 때문에 one to one의 관계이다.
   2. 배송예정일은 주문을 하는 동시에 생성되기 때문에 not null이다.
   3. 배송 완료일은 아직 완료되지 않은 배송이 있을 수 있기 때문에 null을 허용한다.

3. ODBC Implementation.

먼저, 데이터를 6월 18일날 실행했다고 가정하고 만들었기에 오늘의 날짜를 2022년 6월 18일로 가정한다.

1. Initialization()

Initialization단계에서 sql과 연동하고, 20170024.txt파일을 읽어 데이터베이스를 셋팅한다.

내가 설계한 데이터베이스는 기본적으로 여러개의 제품을 동시에 구매하더라도 구매기록이 한번에 한 제품씩 구매한 것으로 기록된다. 따라서 패키지 상품을 구매했을 경우에도 패키지를 패키지의 구성요소들로 분해하여 주문을 넣는다.
package_decomposite()함수가 이미 들어가있는 주문을 decomposite해주는 역할을 한다.

가령 phone_package 이 iPhone 하나 airpot 하나로 구성되어있을 경우,
Product_ID가 phone_packge인 주문이
iPhone
airpot
이렇게 두개의 주문으로 분해된다.

또한, 휴지_패키지가 휴지 3개로 구성되어있다면,
product_ID가 휴지_패키지인 주문의 경우
휴지
휴지
휴지
이렇게 3개의 주문으로 분해된다.

2. Execute()
Execute 단계에서 과제에서 요구하는 query들을 구현했다.
1. Type 1
현재 배송이 완료되지 않은 배달의 목록을 표시해주고, 현재 배송이 완료되지 않은 배달중 사고를 내고싶은 배송번호를 고른다.

번호를 고르면 선택된 배송이 망가지고 해당 배송을 주문한 고객의 연락처 정보를 표시해준다.

그후, 주문을 새로 넣고 새로운 배송정보를 표시해준다.

새로운 주문을 넣을 경우, 기존에 패키지였던 상품이더라도, payment단계에서 각 구성요소들로 분할됐기 때문에, 그 구성요소들을 그대로 다시 주문할 수 있다.
2. Type 2
작년의 payment를 customer_ID를 기준으로 group by를 수행 후, sum(price)값을 구한후, sum(price)를 기준으로 가장 많이 구매한 고객의 ID를 가져온다.

작년에 그 고객이 산 payment들을 product_ID를 기준으로 group by를 수행한후, sum(price)를 구해 그 값을 기준으로 그 고객이 가장 많이 구매한 제품정보를 가져온다.
3. Type 3
작년의 payment에서 product_ID만을 distinct하게 가져와서 보여준다.
이 단계에서 작년에 팔린 총 제품이 몇개인지를 cnt에 저장해둔다.

작년의 payment에서 product_ID를 기준으로 group by를 하고 product와 join을해서 sum(price)를 구하고, 그 값을 기준으로 정렬한다.

type_3_1(cnt);
위와 같은 방법으로 작년의 판매액을 기준으로 상위 cnt개 제품명을 표기한다.
type_3_2(cat);
위와 같은 방법으로 작년의 판매액을 기준으로 상위 cnt/10개 제품명을 표기한다.
4. Type 4
작년의 payment에서 product_ID만을 distinct하게 가져온다.
이 단계에서 작년에 팔린 총 제품이 몇개인지를 cnt에 저장해둔다.

작년의 payment에서 product_ID를 기준으로 group by를 하고 count를 해서 몇개가 팔렸는지 확인한다.
확인 방법은 다음과 같다
이번에 설계한 데이터베이스는 한 payment tuple에 하나의 제품만 구매 가능하다.

가령, 휴지 3개를 한번에 구매를 해도
휴지 1개
휴지 1개
휴지 1개
이런식으로 3번에 나눠서 payment tuple이 생성된다.

또한, initialization단계에서 decomposite()을 통해 package제품을 각각의 구성요소별로 주문을 다시넣어 쪼개놓았다. 쪼갤 시, 고객ID, 구매점포, 구매일시, 배송일시 등의 정보는 기존의 정보와 동일하게 설정해놓고, product_ID만 각 구성요소들로 바꿔서 추가했다.

이렇게 해서 작년에 어떤 제품들이 팔렸고, 각 제품들이 몇번씩 팔렸는지에 대한 정보를 보여준다.

type_4_1(cnt);
위와 같은 방법으로 작년에 판매한 개수를 기준으로 정렬해서 상위 cnt개 제품명을 표기한다.
type_4_2(cnt);
위와 같은 방법으로 작년에 판매한 개수를 기준으로 정렬해서 상위 cnt/10개 제품명을 표기한다

5. Type 5
   주소가 캘리포니아인 점포에 있는 모든 제품의 목록을 나열한 후, 해당 제품을 제외한 결과값들을 출력해준다.
6. Type 6
   delivery 내에서 예정일보다 완료일이 느린 모든 튜플들을 표기한다.
7. Type 7
   A: account정보가 null이 아닌 고객의 ID를 나열한다
   B: 고객 ID가 A 안에 있는 지난달의 payment들을 고객_ID로 group by한 후, sum(price)를 구한다.
   결과: 각 고객의 sum(price)정보를 표기한다.

3. Quit()
기존에는 drop.txt파일을 읽게해서 모든 테이블을 제거해주도록 프로그램을 작성하였다.
그러나, 20170024.txt이외의 텍스트를 넣어도 되는지 명확하지 않았기에, c언어 내에서 각각 drop table을 수행시켰다.

모든 테이블목록을 데이터베이스에서 읽어와서 그대로 제거하려 했으나, foreign key등의 이유로 그렇게 제거하는 데 실패하고, 결국 제거해야하는 순서에 맞게 drop테이블을 각각 작성해주었다.

또한 mysql은 내부의 튜플을 제거하지 않아도 테이블을 드랍시키면 다른 튜플들이 다 드랍되는 것을 보인다.
따라서, 튜플의 제거 없이 바로 테이블을 드랍시켜주었다.

모든 테이블을 드랍한 후에 mysql과의 connection을 끊어준다.
이후 프로그램을 종료한다.