[DB Project 2]

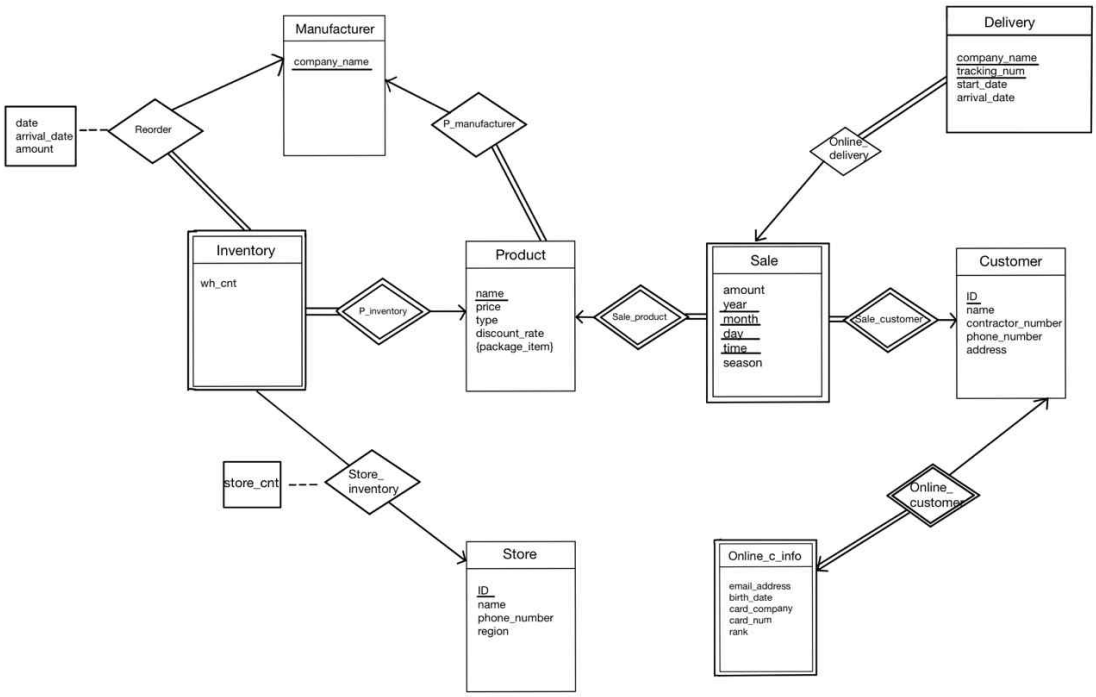
20200376 김하늘

1. 진행 순서

기존의 E-R Schema에서 필요한 부분을 보완한 것을 Logical Schema로 도출해내는 과정에 대해 보이고, 이 Logical schema 가 BCNF 임을 나타내도록 한다. 다음으로 Physical schema diagram에 대한 설명과, Visual studio에서 MYSQL을 사용해 작성한 코드에 대해 설명하도록 한다.

2. E-R Schema 보완 및 Logical Schema 도출

기존에 작성했던 E-R Schema Diagram은 다음과 같다.



수정된 7개의 Entity에 대한 보완점을 설명하도록 한다.

(1) product

기존 product 에는 discount_rate 라는 속성이 존재했지만, 이를 제거하였다. 왜냐하면 상품에 대한 할인율 적용이 아닌 패키지에 대한 할인율 적용이 더 적합하다고 판단을 내렸기 때

문이다.

(2) package

기존에는 package 라는 Entity 가 존재하지 않고, product의 다중 속성으로 package_item 이 존재했었다. 하지만 product 와 package 간에 할인율 속성도 필요하고, 데이터간 중복을 피하기 위해서는 패키지를 별도의 id로 관리하는 것도 필요했다. 또한, 상품과 패키지의 관계도 존재하기에 다중 속성보다는 패키지를 별도의 entity로 분리해주는 것이 더욱 적합하다 판단했다. 패키지와 상품 간에는 package_item을 나타내는 다대다 관계가 생기게 된다. 이는 하나의 패키지가 여러 상품을 가질 수 있고, 마찬가지로 하나의 상품 역시 여러 가지의 패키지에 속할 수 있기 때문이다.

(3) reorder

원래 reorder entity의 주 키는 p_name 이었으나, 같은 상품에 대해 여러번 재주문을 진행할 수 있고 이들을 한 눈에 보기 위해서 주 키를 order_number 라는 새로운 속성으로 정의하였다.

(5) online_c_info

기존에는 온라인 고객 정보에 카드 회사명을 포함하고 있었으나 카드 회사명의 경우 고객이 택하는 것이 더욱 편리하고, 이에 대한 카드 번호만을 보여줌이 적절하다고 생각되어 카드 회사명을 제거했다.

(6) sale

원래 sale entity의 주 키가 매우 복잡했기에, sale_number라는 새로운 속성을 추가해 sale의 주 키가 되도록 했다. 또한, 판매에 대한 년, 월, 일 정보를 따로 저장하지 않고 datetime domain을 갖는 sale_date 속성으로 축약해서 나타내었다. 날짜로부터 도출 가능한 정보인 season 속성은 삭제하였다. 추가로 새로 생긴 package entity에 대한 판매도 잡아내야하므로 외래키로 패키지 id를 갖도록 했다. 끝으로 offline 이라는 int 형 속성을 두어서 null 값이면 온라인 판매를 나타내고, s_id 값이면 구매가 이루어진 오프라인 가게명을 저장하도록 했다.

(7) delivery

기존에는 배송에 대한 정보를 나타내기 위해 p_name을 외래키로 참조했다면, 이제는 sale_number를 이용해 해당 정보에 접근 가능하도록 했다. 이렇게 한 이유는 sale의 주 키는 sale_number이기 때문이다. 또한 예정 배송 도착일을 나타내는 promise_date를 추가하였는데 이는 각 배송마다 예송 도착일에 차이가 있고 이로써 미배달 상품에 대한 조회가 수월해지기 때문이다.

앞서 설명한 보완점을 바탕으로 E-R schema를 Logical schema로 나타내면 다음과 같다.

◇ Product (name, company_name, price, type)
◇ Manufacturer (company_name)
◇ Package (pkg_id, pkg_name, discount_rate)
◇ Package_item (pkg_id, p_name)
◇ Inventory (p_name, wh_cnt)
◇ Reorder (order_number, p_name, company_name, amount, order_date, arrival_date)
◇ Store (s_id, name, phone_number, region)
◇ Store_inventory (s_id, p_name, store_cnt)
◇ Customer (c_id, name, contractor_number, phone_number, address)
◇ Online_c_info (c_id, email, birth_date, card_number, ranking)
◇ Sale (sale_number, c_id, p_name, pkg_id, sale_date, offline, amount)
◇ Delivery (d_company_name, tracking_num, sale_number, start_date, promise_date, arrival_date)

다대다 관계를 나태내는 package_item 에 대한 별도의 schema를 작성해주었다.
reorder 의 경우 manufacturer와 inventory의 일대다관계이기 때문에 inventory 측에 reorder의 모든 속성을 넣어줘도 되지만, 데이터 중복을 피하기 위해 별도의 schema를 작성하였다.
마찬가지로 store_inventory 역시 store와 inventory간 일대다의 관계를 맺고 있지만 데이터의 중복을 피해주기 위해 별도의 schema로 분리해서 나타내었다.

3. Logical Schema의 BCNF 검사

앞서 작성한 Logical schema가 BCNF를 만족하는지 확인하기 위해서는 각 Schema에 대한 functional dependency를 파악해야만 한다.

(1) Product (name, company_name, price, type)
F = { name -> company_name, price, type }
name의 closure 는 Relation product의 모든 속성을 포함하므로 product은 BCNF를 만족한다.

(2) Manufacturer (company_name)
F = { company_name -> company_name }
company_name의 closure 는 Relation manufacturer의 모든 속성을 포함하므로 manufacturer은 BCNF를 만족한다.

(3) Package (pkg_id, pkg_name, discount_rate)

F = { pkg_id -> pkg_name, discount_rate }

pkg_id의 closure 는 Relation package의 모든 속성을 포함하므로 package는 BCNF를 만족한다.

(4) Package_item (pkg_id, p_name)

F = { pkg_id, p_name -> pkg_id, p_name }

pkg_id, p_name의 closure 는 Relation package_item의 모든 속성을 포함하므로 package_tiem은 BCNF를 만족한다.

(5) Inventory (p_name, wh_cnt)

F = { p_name -> wh_cnt }

p_name 의 closure 는 Relation inventory 의 모든 속성을 포함하므로 inventory는 BCNF를 만족한다.

(6) Reorder (order_number, p_name, company_name, amount, order_date, arrival_date)

F = { order_number -> p_name, company_name, amount, order_date,
p_name -> company_name }

order_number의 closure 는 Relation reorder의 모든 속성을 포함하므로 해당 functional dependency는 BCNF를 위반하지 않는다.

p_name의 closure는 Relation reorder의 모든 속성을 포함하지 못하므로, 해당 functional dependency는 BCNF를 위반하게 된다. 따라서 해당 schema를 쪼개면 다음과 같다.

-> reorder (order_number, p_name, amount, order_date, arrival_date)

-> reorder2 (p_name, company_name)

쪼갠 두 relation 은 모두 BCNF를 만족하게 된다.

reorder2의 경우 product에 포함되기 때문에 별도의 table을 만들지는 않고, reorder에 company_name 속성을 제거하도록 한다.

(7) Store (s_id, name, phone_number, region)

F = { s_id -> name, phone_number, region }

s_id 의 closure 는 Relation store 의 모든 속성을 포함하므로 store는 BCNF를 만족한다.

(8) Store_inventory (s_id, p_name, store_cnt)

F = { s_id, p_name -> store_cnt }

s_id, p_name 의 closure 는 Relation store_inventory 의 모든 속성을 포함하므로 store_inventory는 BCNF를 만족한다.

(9) Customer (c_id, name, contractor_number, phone_number, address)
F = { c_id -> name, contractor_number, phone_number, address,
contractor_number -> c_id, name, phone_number, address }

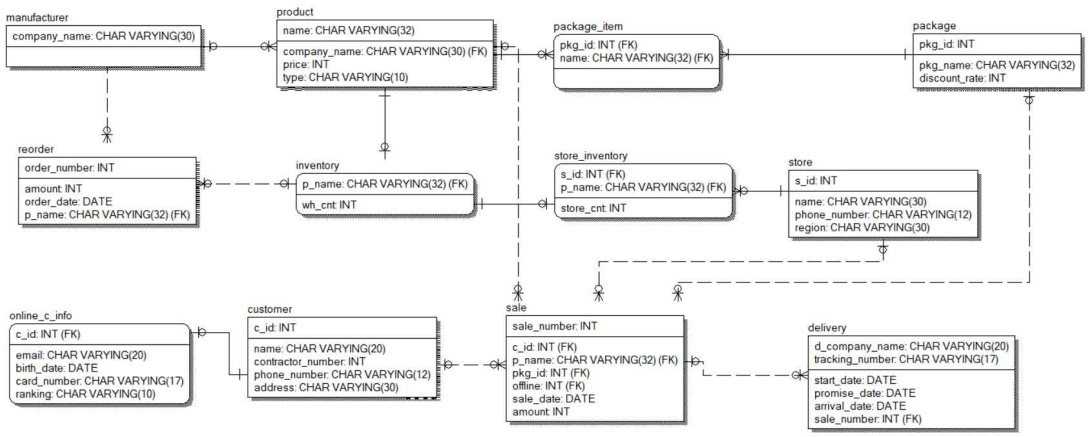
c_id 의 closure 는 Relation customer 의 모든 속성을 포함하므로 해당 functional dependency는 BCNF를 위반하지 않는다.
contractor_number 의 closure 는 Relation customer 의 모든 속성을 포함하므로 해당 functional dependency는 BCNF를 위반하지 않는다.
따라서 customer 는 BCNF를 만족한다.

(10) Online_c_info (c_id, email, birth_date, card_number, ranking)
F = { c_id -> email, birth_date, card_number, ranking }
c_id 의 closure 는 Relation online_c_info 의 모든 속성을 포함하므로 online_c_info 는 BCNF를 만족한다.

(11) Sale (sale_number, c_id, p_name, pkg_id, sale_date, offline, amount)
F = { sale_number -> c_id, p_name, pkg_id, sale_date, offline, amount }
sale_number 의 closure 는 Relation sale 의 모든 속성을 포함하므로 sale은 BCNF를 만족한다.

(12) Delivery (d_company_name, tracking_numer, sale_number, start_date, promise_date, arrival_date)
F = { d_company_name, tracking_number -> sale_number, start_date, promise_date, arrival_date }
d_company_name, tracking_number 의 closure 는 Relation delivery 의 모든 속성을 포함하므로 delivery는 BCNF를 만족한다.

4. Physical Schema Diagram과 설명



Physical Schema Diagram 은 앞서 BCNF로 정규화한 최종 Logical Schema Diagram을 erwim 프로그램으로 구현한 것이다. 따라서 logical schema 와 완전히 동일하다고 볼 수 있다. 추가로 table 간 관계가 어떻게 이루어져 있는지 설명하도록 한다.

(1) Product (name, company_name, price, type)
product 의 name은 최대 32개의 문자를 사용할 수 있도록 했고, 주 키를 name으로 설정해서 모든 제품명이 상이하고 또 이 상이한 제품명으로 모든 제품을 구분하도록 했다.
왜냐하면 이후 쿼리를 고려했을 때 product name을 바로 알 수 있는 편이 join을 줄일 수 있게 한다고 생각했기 때문이다. price 는 정수값으로 주고, type은 최대 10개의 문자를 갖도록 했다. product 은 manufacturer 와 일대다의 관계를 갖게 되므로 product은 one 측인 manufacturer의 주 키를 속성으로 갖게 된다.

(2) Manufacturer (company_name)
manufacturer 는 company_name 이라는 속성을 갖고 최대 30개의 문자를 사용해 나타낼 수 있도록 하였다.

(3) Package (pkg_id, pkg_name, discount_rate)
package 는 정수형인 id로 모든 패키지를 구분하도록 했고 패키지 name 은 최대 32개의 문자를 갖으며, 할인율(%)은 정수로 나타내도록 했다.

(4) Package_item (pkg_id, p_name)
package_item 은 패키지와 상품 간의 다대다 관계를 나타내기 위한 identifying table로 둘의 주 키를 자신의 주 키로 갖게 된다.

(5) Inventory (p_name, wh_cnt)
inventory는 각 제품에 재고를 나타내야 하므로, 제품과 일대일의 관계를 갖게 되고 동시에 제품의 주 키가 있어야 재고를 나타낼 수 있으므로 inventory 는 product을 identify 하게 된다. 따라서 inventory는 product의 주 키인 name을 자신의 주 키로 갖고 추가로 이에 대한 재고수를 정수형으로 나타낸다.

(6) Reorder (order_number, p_name, amount, order_date, arrival_date)
reorder는 주문번호, 수량, 주문날짜, 회사명을 자신을 속성으로 갖는데 각각은 정수형, 중수형, datetime형, 문자형이다. 제품명을 이용해 product table과 결합해 manufacturer을 알아낼 수 있다.

(7) Store (s_id, name, phone_number, region)
store는 정수형인 s_id로 모든 가게를 구분하도록 했으며, 나머지 각각의 속성인 가게명, 전화번호, 지역에는 최대 문자열 길이를 설정해 나타내었다.

(8) Store_inventory (s_id, p_name, store_cnt)

store inventory 는 가게와 재고 table의 관계를 나타내며, 각각 일대일의 관계를 맺는다. 추가로 가게 재고 개수에 대한 자신을 속성을 정수형으로 갖는다. 재고와 가게 재고를 분리해서 저정함으로써 각각의 따로 관리하려 했다.

(9) Customer (c_id, name, contractor_number, phone_number, address)
customer 는 id로 모든 고객을 구분가능하게 했으며, 이름, 계약번호, 휴대폰번호, 주소를 저장하도록 했다. 이때 계약번호는 계약한 고객들에 한해서 부여되도록 했다. 이로 인해 불가피하게 null 값을 갖는 것은 감수해야한다.

(10) Online_c_info (c_id, email, birth_date, card_number, ranking)
온라인 고객에 한해서 이메일, 출생일, 카드번호, 등급을 저장하도록 했고 이는 고객 id 가 있어야 분별이 가능하므로 customer를 identify 하도록 했다.

(11) Sale (sale_number, c_id, p_name, pkg_id, sale_date, offline, amount)
sale 은 자신의 판매 번호로 모든 속성을 구분하고 있지만, 외래키가 많이 필요하다. 우선 상품이랑 패키지에 대한 판매가 생길 수 있으므로 product table의 name과 package table의 pkg_id를 속성으로 갖는다. 더불어서 고객 id를 나타내야하므로 customer table의 주 키를 속성으로 가져온다. 추가로 판매 날짜, 수량, offline을 저장하는데, offline의 값이 null 이면 온라인 구매 고객이므로 배송을 발생시켜야하고, null 이 아닌 정수값이 있으면 이는 오프라인 가게 번호를 나타내도록 했다.

(12) Delivery (d_company_name, tracking_num, sale_number, start_date, promise_date, arrival_date)
배송은 판매에 대해 발생하므로 판매 table의 주 키를 자신을 속성으로 갖도록 해서 어떤 판매에 대한 배송인지 구분하도록 했다. 이때 분할 배송도 가능하므로 판매번호가 다른 속성을 구분해주지는 못하고 배송 업체와 배송번호가 배송을 개별적으로 구분하게 한다.

5. MYSQL 구현에 대한 코드 설명

mysql 코드 작성을 위해 실제 사고했던 순서대로 작성을 하도록 한다.

1. TYPE1

- TYPE1 : Find the contact information for the customer
select * from customer as c, sale as s where c.c_id = s.c_id and s.sale_number in (select sale_number from delivery where tracking_number='%s' and d_company_name = 'USPS'), track

우선 배송 table에서 입력받은 추적 번호 k와 배송업체 'USPS'에 대한 판매 번호를 찾는다.

다음으로 판매 table 과 고객 table을 join 해서, 판매 번호가 방금 찾은 판매번호와 동일한

경우하고, 해당 판매 table의 고객 id와 고객 table의 고객 id가 같은 고객에 대한 모든 정보를 출력하도록 했다.

- TYPE1-1 : Find the contents of that shipment and create a new shipment
select p.pkg_name, pi.p_name, amount from sale as s, package as p, package_item as pi where pi.pkg_id=p.pkg_id and s.pkg_id = p.pkg_id and s.sale_number in (select sale_number from delivery where tracking_number='%s' and d_company_name = 'USPS')", track

우선 배송 table에서 입력받은 추적 번호 k와 배송업체 'USPS'에 대한 판매 번호를 찾는다.

다음으로 판매 table과, 패키지 table, 패키지아이템 table을 join 해서, 패키지명과 패키지에 해당하는 제품명, 수량을 출력하도록 한다. 즉, 방금 찾은 판매 번호를 갖는 판매에 대해 패키지 id가 패키지 table의 id와 일치한 것을 찾고, 그 패키지 id와 일치하는 패키지아이템을 찾는 것이다.

update inventory set wh_cnt = wh_cnt - %d where p_name='%s'", tmp, temp2[j]

파손되어 도착하지 못한 배송에 대해 새로운 배송을 만들기에 앞서 이는 주문 없는 재고 감소와 같으므로 재고 table에서 패키지에 해당하는 제품들의 수를 수량만큼 감소시켜주었다.

insert into delivery values ('USPS', '0000000000000007', %d, curdate(), adddate(curdate(), 7), null)", tmp2

다음으로 새로운 배송을 만들어야하기에, 같은 배송업체가 그 다음 tracking_number를 붙여서, 기존과 동일한 판매번호에 대해 배송시작일은 현재로 하고, 배송 예정일은 현재에서 7일 뒤로 설정하였다. 이를 delivery table 에 insert 해줌으로써 배송이 반영되도록 했다.

2. TYPE2

- TYPE2 : Customer who has bought the most (in price) in the past year

create view p_price as with p_price(c_id, p_name, amount, price, final_price) as (select c_id, p_name, amount, p.price, amount * p.price from sale, product as p where p.name = sale.p_name and sale_date like '2021%%' order by c_id) select c_id, sum(final_price) as final_price from p_price group by c_id order by c_id

작년의 제품 판매와 그 판매에 대한 가격을 나타내는 뷰를 생성한다.

우선 with를 사용해 판매 table과 제품 table를 join 하고, 판매 제품명과 제품 제품명이 같고, 판매가 작년에 이루어졌을 때를 찾아낸다. 다음으로 이 임시 table을 활용해서 고객 id 별

로 그룹을 묶고 고객의 id와 총 지불 금액을 출력한다.

create view pkg_price as with pkg_list(pkg_id, p_name, price, discount_rate) as (select p.pkg_id, pi.p_name, pd.price, p.discount_rate from package as p, package_item as pi, product as pd where pd.name=pi.p_name and p.pkg_id=pi.pkg_id) select pkg_id, sum(price) as price, discount_rate, sum(price)*(100-discount_rate)*0.01 as final_price from pkg_list as pl group by pl.pkg_id

패키지 판매와 그 패키지 판매에 대한 가격 뷰를 생성하기에 앞서, 패키지와 그 패키지에 할인율을 적용한 패키지 가격을 나타내는 뷰를 생성해야한다.

따라서 with를 사용해서, 패키지와 패키지 아이템, 제품을 join 한 다음, 패키지 table의 패키지 id와 패키지 아이템의 패키지 id가 같고, 그 패키지 id를 갖는 제품들을 추려서, 패키지 id와 각 제품명, 제품 가격, 할인율을 출력한다. 이 임시 table을 이용해서 패키지 id 별로 그룹을 만들고, 패키지 id, 총 제품 가격, 할인율, 총 제품 가격*할인율을 출력하도록 한다. 즉 최종적으로 알고 싶은건 할인을 적용한 패키지의 가격인 것이다.

create view pk_price as with pk_price(c_id, pkg_id, amount, price, final_price) as (select c_id, pk.pkg_id, amount, pk.final_price, amount*pk.final_price from sale, pkg_price as pk where pk.pkg_id=sale.pkg_id and sale_date like '2021%%' order by c_id) select c_id, sum(final_price) as final_price from pk_price group by c_id order by c_id

작년의 패키지 판매와 그 패키지 판매에 대한 가격을 나타내는 뷰를 생성한다.
우선 with를 사용해 판매 table과 패키지 가격 table를 join 하고, 판매 패키지 id와 패키지 가격의 패키지 id가 같고, 판매가 작년에 이루어졌을 때를 찾아낸다. 다음으로 이 임시 table을 활용해서 고객 id 별로 그룹을 묶고 고객의 id와 패키지에 대한 총 지불 금액을 출력한다.

create view type2 as select * from p_price union select * from pk_price

위에서 만든 두 뷰인 작년 판매 제품과 작년 판매 패키지를 합친다.

with type2_max(c_id, max_price) as (select c_id, sum(final_price) from type2 group by c_id order by c_id) select c_id, max_price from type2_max where max_price >= (select max(max_price) from type2_max)")

가장 구매 금액이 높은 고객은 산출한다.

with를 사용해서 합친 뷰인 type2를 고객 id로 묶고, 고객별 제품 및 패키지 구매에 대한 총 구매 금액을 나타내고, 이 임시 table을 사용해서 가장 높은 금액을 지불한 고객 id와 그 금

액을 출력한다.

- TYPE2-1 : The product that the customer bought the most

create view type2_1 as
select p_name, amount from sale where sale_date like '2021%%' and c_id=%d and p_name is not null union
select p.p_name, s.amount from sale as s, package_item as p where s.pkg_id=p.pkg_id and sale_date like '2021%%' and c_id=%d and s.pkg_id is not null", tmp, tmp

위에서 찾은 가장 높은 금액을 지불한 고객이 구매한 제품 리스트에 대한 뷰를 생성한다.

* 판매 table에서 작년에 판매되고 해당 고객이 구매한 제품의 제품명, 수량 선택
* 판매 table 과 패키지 아이템 table을 join 하고, 해당 고객이 작년에 구매한 패키지의 id에 해당하는 제품명과 수량 선택

이 두 선택(제품, 패키지 제품)을 합해서 type2 뷰를 생성한다.

select p_name, amount from type2_1 group by p_name having amount >= max(amount) order by amount DESC

앞서 만든 type2 뷰를 수량 순으로 정렬하고, 수량이 가장 큰 제품명과 수량을 출력하도록 한다.

3. TYPE3

- TYPE3 : Find all products sold in the past year
create view t3_1 as with t3_1(p_name, amount, price) as
(select p_name, amount, price from sale, product as p where sale.p_name=p.name and sale_date like '2021%%' and p_name is not null) select p_name, amount, amount*price as price from t3_1

작년에 판매된 제품에 대해 뷰를 생성한다.

with 사용해서 판매 table과 product table을 합쳐서 작년에 판매된 제품 중 제품명, 수량, 가격을 추려내고 이 임시 table에 대해 제품명, 총 제품 수량, 총 수량과 가격을 곱한 최종 금액을 출력한다.

create view t3_4 as select p_name, sum(amount) as amount, sum(price) as price

from t3_1 group by p_name

작년에 판매된 제품을 제품명으로 그룹지은 뷰를 생성한다.

create view t3_2 as with t3_2(p_name, amount, price) as
(select p.p_name, s.amount, price from sale as s, package_item as p, product as pd where p.p_name=pd.name and s.pkg_id=p.pkg_id and sale_date like '2021%%' and s.pkg_id is not null) select p_name, amount, amount*price as price from t3_2“

작년에 판매된 패키지의 제품에 대해 뷰를 생성한다.

with 사용해서 판매 table, 패키지 아이템 table, 제품 table을 join 한 후, 작년에 판매된 패키지에 대한 제품명, 수량, 가격을 추려낸 다음, 이 임시 table을 사용해서 제품명, 총 제품 수량, 총 수량과 가격을 곱한 최종 금액을 출력한다.

create view t3_5 as select p_name, sum(amount) as amount, sum(price) as price from t3_2 group by p_name

작년에 판매된 패키지 내의 제품을 제품명으로 그룹지은 뷰를 생성한다.

create view t3_3 as select * from t3_4 union select * from t3_5

작년에 판매된 제품과 패키지 내 제품을 합쳐서 완전한 판매 제품 뷰를 생성한다.

select p_name, sum(amount) as amount, sum(price) as price from t3_3 group by p_name order by price desc

최종적으로 완전한 판매 제품 뷰를 가격순이 높은 순으로 정렬해서 제품명과, 제품 수량, 제품 가겨을 출력한다.

- TYPE3-1 : Find the top k products by dollar-amount-sold

select p_name, sum(amount) as amount, sum(price) as price from t3_3 group by p_name order by price desc

사용자로부터 k개를 입력받아서 가격 높은 순대로 k 개의 제품을 출력한다. k개를 출력하기 위해서는 출력개수를 세서 k개가 되면 break 하도록 한다.

- TYPE3-2 : Find the top 10%% products by dollar-amount-sold

select p_name, sum(amount) as amount, sum(price) as price from t3_3 group by

p_name order by price desc

상위 10% 상품 개수를 찾아서 그 만큼 제품을 출력한다.

4. TYPE4

- TYPE4 : Find all products by unit sales in the past year

select p_name, sum(amount) as amount, sum(price) as price from t3_3 group by p_name order by amount desc

최종적으로 완전한 판매 제품 뷰를 판매량이 높은 순으로 정렬해서 제품명과, 제품 수량, 제품 가겨을 출력한다.

- TYPE4-1 : Find the top k products by unit sales

select p_name, sum(amount) as amount, sum(price) as price from t3_3 group by p_name order by amount desc

사용자로부터 k개를 입력받아서 판매량이 높은 순대로 k 개의 제품을 출력한다. k개를 출력하기 위해서는 출력개수를 세서 k개가 되면 break 하도록 한다.

- TYPE4-2 : Find the top 10%% products by unit sales

select p_name, sum(amount) as amount, sum(price) as price from t3_3 group by p_name order by amount desc

상위 10% 상품 개수를 찾아서 그 만큼 제품을 출력한다.

5. TYPE5

- TYPE5 : Find those products that are out-of-stock at every store in California

create view type5 as select p_name, count(p_name) as count from store as s, store_inventory as si where store_cnt=0 and s.s_id=si.s_id and region='California' group by p_name

가게 table과 가게 재고 table을 join 한 뒤, 캘리포니아에 위치하는 가게에 대해 재고가 0인 제품을 찾고 제품명별로 그룹을 지어 그 개수를 함께 출력한 뷰를 생성한다.

select * from type5 where count = (select count(s_id) from store where region='California')

앞서 생성한 뷰를 사용해서 재고가 0인 제품의 개수가 캘리포니아에 위치하는 가게 수가 같은 것을 추려낸다.

6. TYPE6 : Find packages that were not delivered within the promised time

select p.pkg_id, p.pkg_name, s.amount from sale as s, package as p
where p.pkg_id = s.pkg_id and s.pkg_id is not null and s.sale_number in
(select sale_number from delivery where promise_date < arrival_date or (promise_date<curdate() and arrival_date is null))

우선 배달 table에서 예상 배송일 보다 늦거나 예상 배송일이 지났는데 도착하지 않은 판매 번호를 추려낸다.

다음으로, 판매 table, 패키지 table을 결합해서 해당 판매 번호를 갖는 패키지 id 와 패키지 명, 수량을 출력해낸다.

7. TYPE7 : Generate the bill for each customer for the past month

create view type7 as with type7_1(c_id, pkg_id, amount, price) as
(select c_id, s.pkg_id, amount, final_price from sale as s, pkg_price as p where s.pkg_id=p.pkg_id and sale_date like '2022-05%%' and c_id in (select c_id from customer where contractor_number is not null))
select c_id, pkg_id as product_pkg, amount, amount*price as price from type7_1

계약 고객의 지난달 구매 패키지에 대한 청구서 뷰를 생성한다.

우선 계약 고객 id를 찾아낸다.
다음으로, with를 사용해서 판매 table, 패키지 가격 table을 결합해서 지난달 계약 고객의 구매에 대한 고객 id, 패키지 id, 수량, 패키지 금액을 추려낸다. 이 추려낸 임시 table을 활용해 고객 id, 패키지 id, 수량, 수량*패키지가격을 출력한다.

create view t7 as with t7_1(c_id, p_name, amount, price) as
(select c_id, p_name, amount, price from sale as s, product as p where p.name=s.p_name and sale_date like '2022-05%%' and c_id in (select c_id from customer where contractor_number is not null))
select c_id, p_name as product_pkg, amount, amount*price as price from t7_1

계약 고객의 지난달 구매 제품에 대한 청구서 뷰를 생성한다.

우선 계약 고객 id를 추려낸다.
다음으로 with를 사용해서 판매 table과 product table을 join 한 후, 해당 고객들이 지난달에 구매한 제품에 대한 고객 id, 제품명, 수량, 가격을 추려낸다. 이 임시 table을 사용해서 고객 id, 제품명, 수량, 수량*제품 가격을 출력한다.

create view type7_2 as select * from t7 union select * from type7

위 두 뷰를 결합해서 최종적인 청구서를 완성한다.

select c_id, sum(price) as total_price from type7_2 group by c_id

최종적 청구서를 고객 id로 묶어서 최종 지불 금액을 출력한다.

select * from type7_2

최종 지불 금액과 더불어 구매 목록 명세를 나타내기 위해 별도로 출력을 낸다.

6. E-R Model picture