목차

1-a. 프로젝트 폴더 계층 구조

설명 : **sg**폴더가 본인이 한것들입니다.

- **src/main/java/**
  - **com/bookmarket/app**
    - **controller : MVC** 컨트롤러.
    - **dto : MVC DTO.**
    - **mapper : MVC DAO.**
    - **restcontroller : API** 용 컨트롤러.
    - **service : MVC** 모델 서비스 레이어.

        - **BookMarketApplication.java :** 프로그램 진입점.
- **src/main/resources/**
    - **mapper : SQL** 코드들.
- **src/main/webapp/**
    - **resources : css, js, Multipart** 등
    - **WEB-INF/views : MVC View (JSP)**

1-b. 사용 기술 및 언어

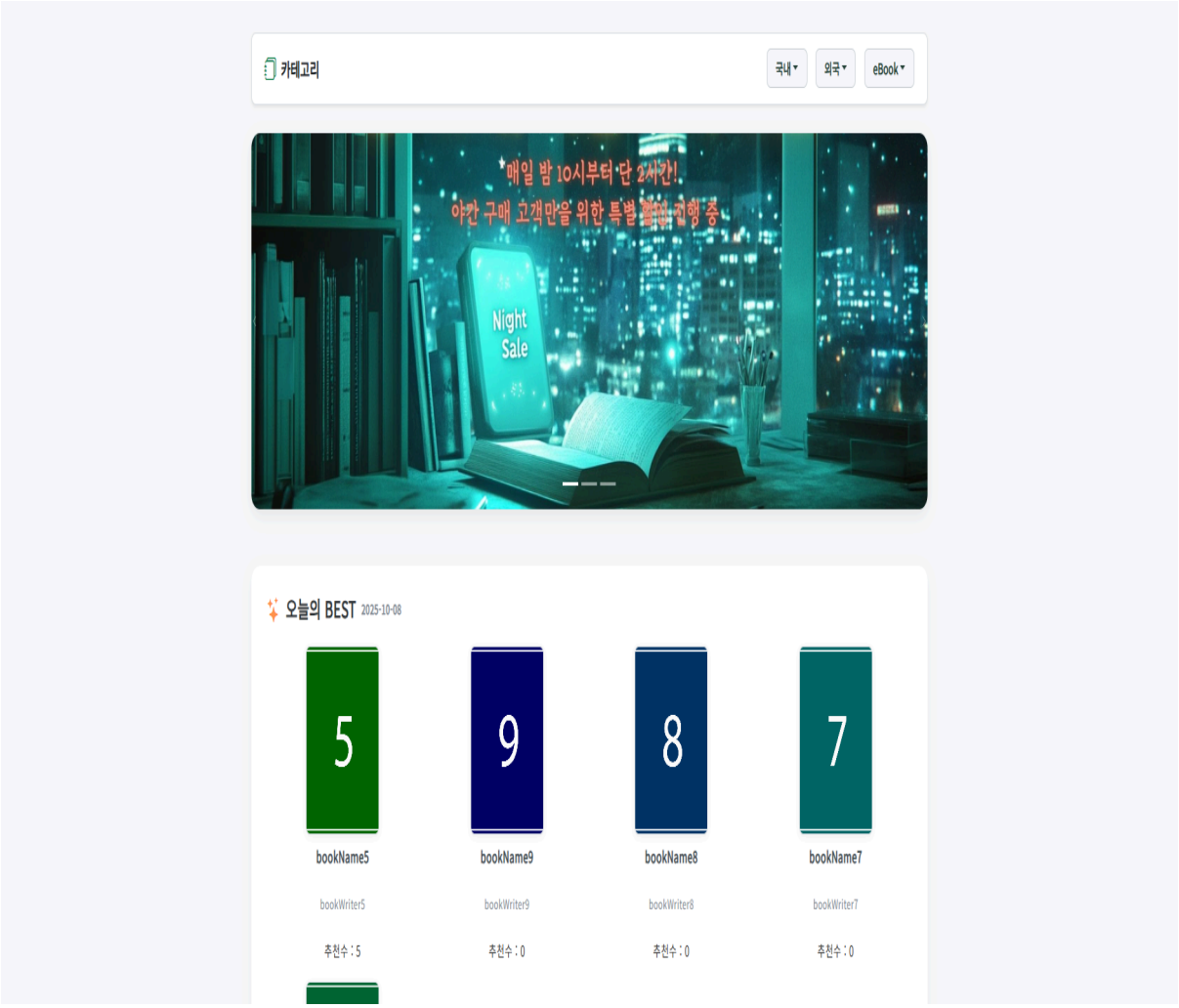
- 기술 **: Spring boot, mybatis, oracle sql, java, ajax, jsp**
- 언어 **: SQL, JAVA, JS**

2-a. ERD

ERD 주소 **:** [**https://www.erdcloud.com/d/CptyXbQ9G42t5ZCfi**](https://www.erdcloud.com/d/CptyXbQ9G42t5ZCfi)

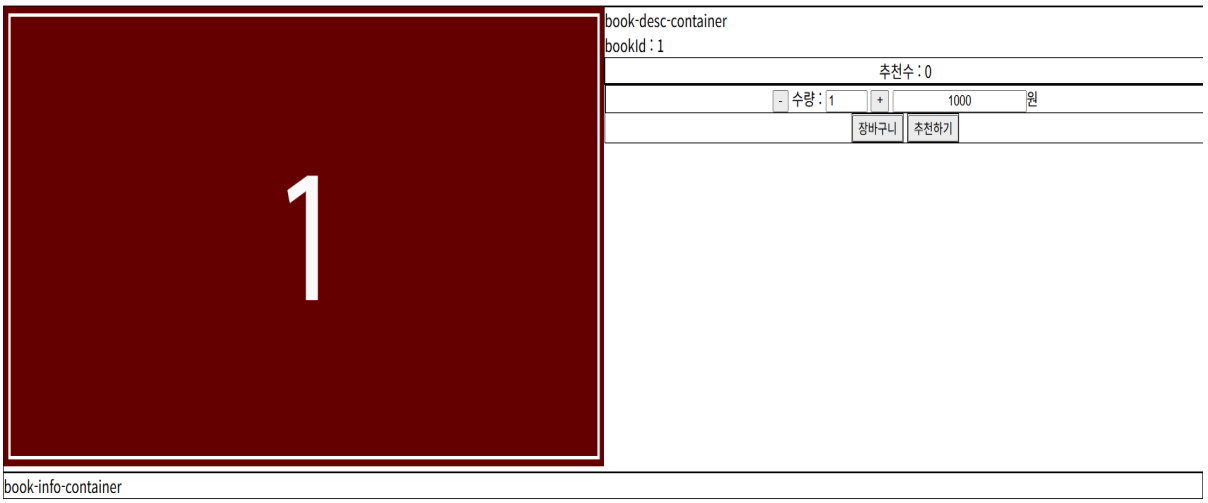
설명 **:** 출판사가 책 등록을 해서 판매하고 사용자가 사는 상황을 고려하여 만들었습니다.

3. 메인페이지



책 추천 설명 : 판매가 가능한 책들 중에서 오늘날짜 - 7 부터 오늘날짜까지 추천수를 기준으로 상위 5개만 가져오도록 하였습니다.

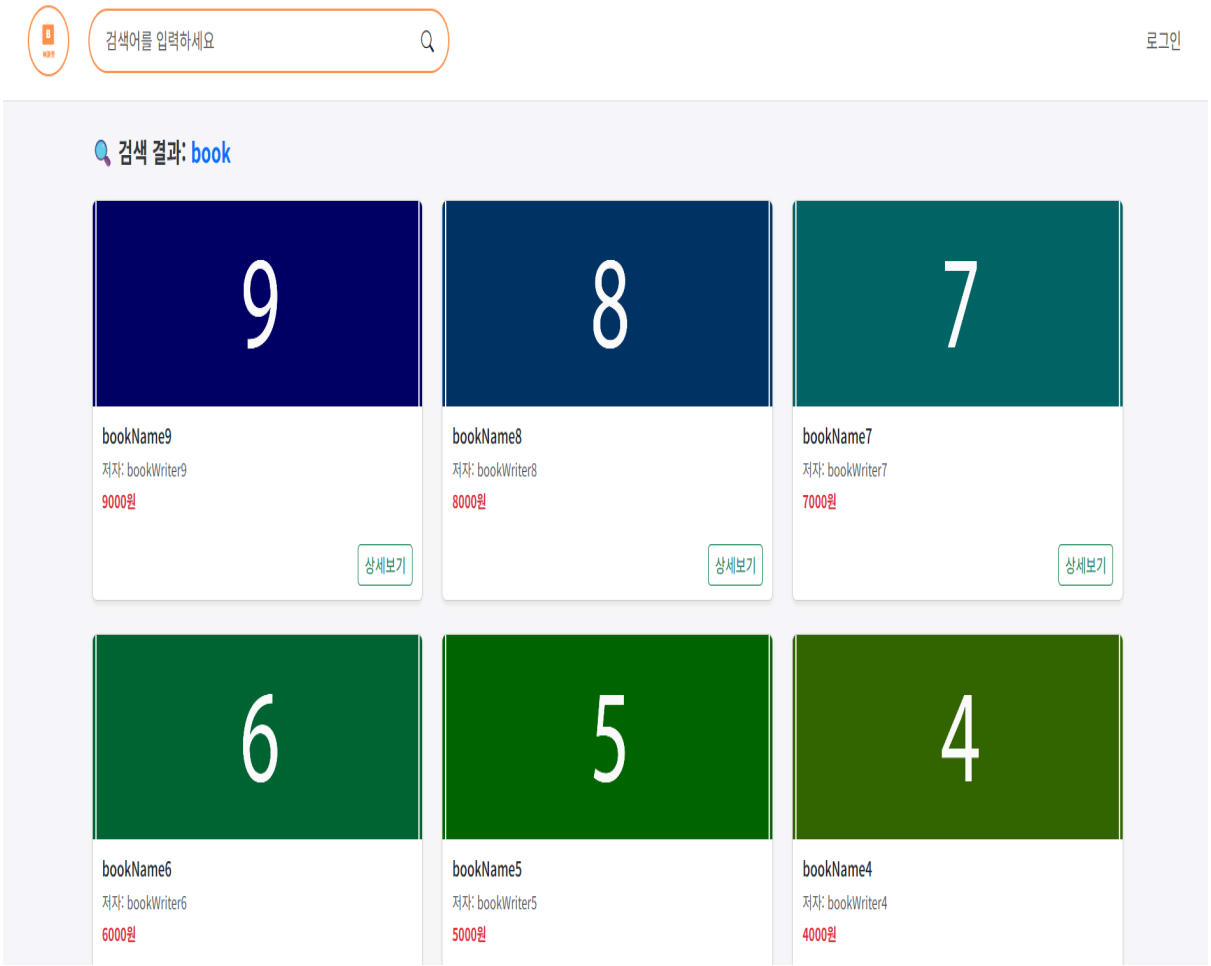
4-a. 책 상세정보



설명 :

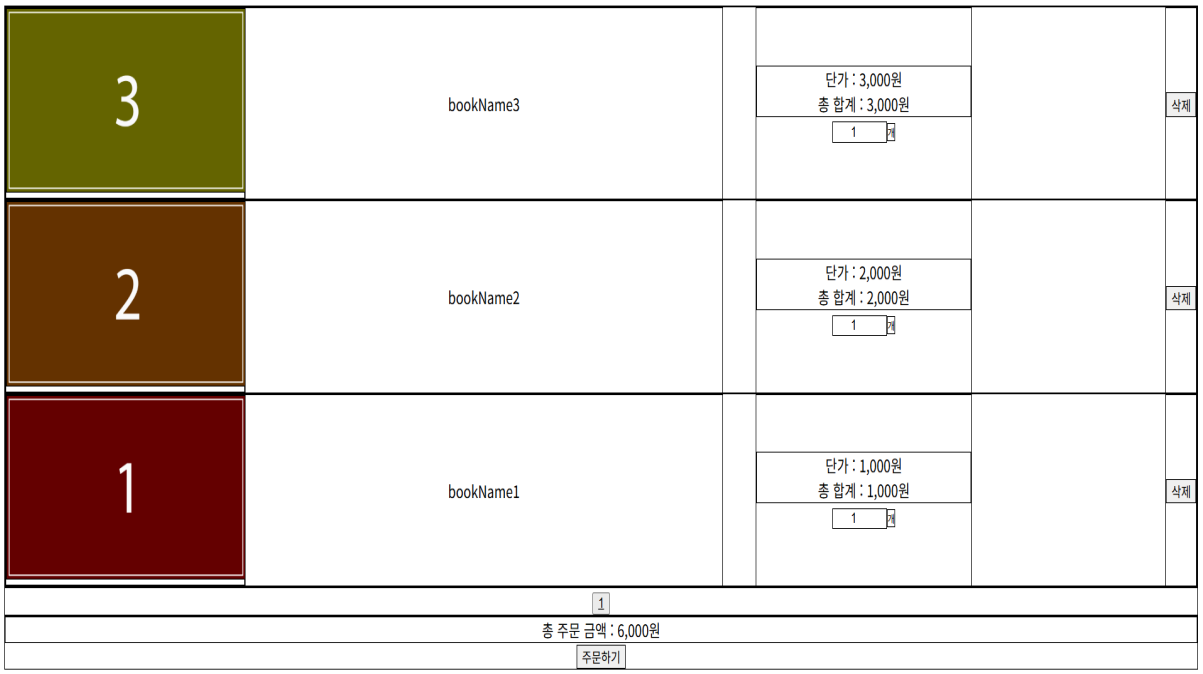
- 수량 표시 : **if**문으로 작성한게 아닌 효율성 및 가독성을 위해 정규식 사용하였습니다. 초기 수량을 **1**로 표시를 해 수량의 변화가 발생하고 해당 변화가 정규식에 부합하면 해당 값으로 표시하고 정규식에 부합하지 않으면 롤백처리를 통해 이전 값을 표시합니다. 이전 값은 정상처리된 값이기 때문에 롤백처리가 됩니다.
- 가격 표시 : 수량 표시와 동일한 원리이며 원화표시 정규식을 사용하였습니다.
- 추천하기 : 추천테이블에서 오늘날짜에 해당 데이터가 있는지 없는지를 확인하여 구현하였습니다. 있으면 실패 없으면 성공입니다. **1**일 **1**회로 제한하였습니다.
- 장바구니 : 물건 넣기 기능입니다. 추천하기 기능과 비슷하게 해당 데이터가 있는지 없는지에 대한 판별을 기준으로 구현하였습니다. 따라서 장바구니에 중복으로 책을 넣을 수가 없습니다.
- 절판표시 : 재고가 **0**이거나 절판되면 책을 장바구니에 넣을 수 없도록 하였습니다.

4-b. 책 검색



설명 : 판매신청된 책들중에서 판매신청완료된 책들을 **WHERE %keyword%** 필터링을 통한뒤, 최신순으로 페이징 처리를 하여 보여주도록 하였습니다. **1**페이지당 **15**개 기준입니다.

5. 장바구니

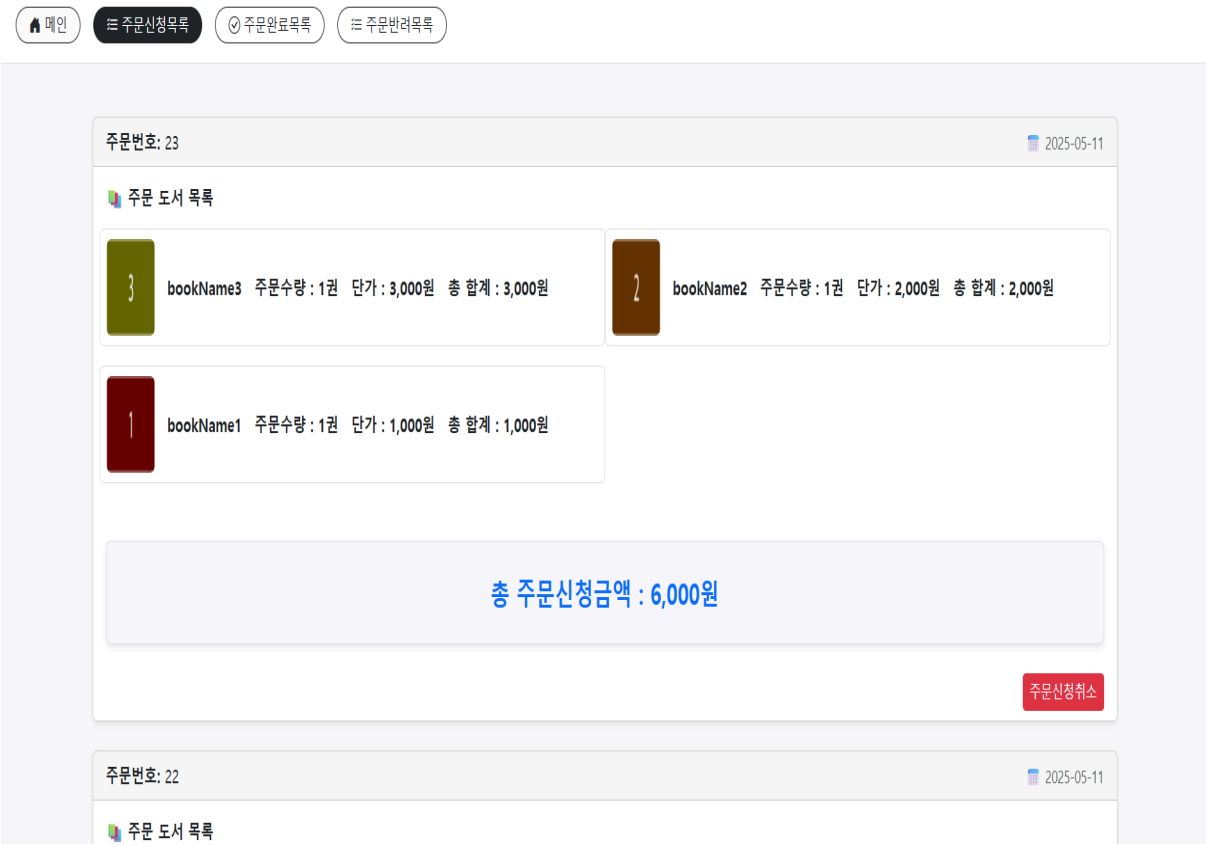


설명 :**1**페이지당 **5**개, 장바구니 물건 최대 **15**개로 제한하였습니다. 총 **3**페이지입니다. 장바구니 페이지에 있는 상품들은 **JS**쪽에서 배열로 저장되어 있습니다.

AJAX 사용해서 동적처리하였습니다. 이벤트 변화에 따른 장바구니 데이터가 서버에 저장이 됩니다. 이벤트가 시작되면 서버로 요청을 해서 실시간으로 데이터를 저장하도록 하였습니다.

- 총 합계 : 개수 올리면 자동으로 변경되게 하였습니다. 이벤트 발생시 배열안에 있는 특정요소를 탐색하여 해당 요소의 총합을 구하게 됩니다.
- 총 주문 금액 : 모든 페이지의 주문 합계가 계산됩니다. **2, 3** 페이지의 가격또한 포함됩니다. 배열을 통한 탐색이므로 뒤쪽 페이지의 대한 금액또한 쉽게 구할 수 있습니다.
- 최대 개수 제한 이유 : 장바구니에 제한없이 계속해서 넣는 것은 이상할 뿐더러 제한을 너무 폭넓게 잡으면 장바구니에 있는 상품을 주문시 너무 많은 입력이 발생하므로 제한을 하였습니다.

6. 주문



주문처리과정 : 장바구니에서 주문을 하면 주문데이터를 입력하게 되는데 주문데이터를 입력시 주문과 주문상세를 **INSERT ALL**을 이용해서 하나의 트랜잭션으로 묶어서 처리하였습니다. 이렇게 하나로 묶지 않고 따로 처리하게 되면 주문과 주문상세의 데이터가 일치하지 않는 경우가 발생할 수 있기에 이렇게 묶어서 처리하였습니다.

주문확인 : 주문한 것을 확인할 때마다 테이블에 총주문금액 컬럼을 추가하지 않아도 주문에 있는 상품의 총합을 구하는 연산을 통해 구할 수 있습니다. 하지만 이렇게 하면 주문을 확인할 때마다 연산을 해야하는 비효율적인 과정이 생기게 됩니다. 따라서 주문 테이블에 총주문금액 컬럼을 추가하여 이러한 연산과정을 없애보았습니다.

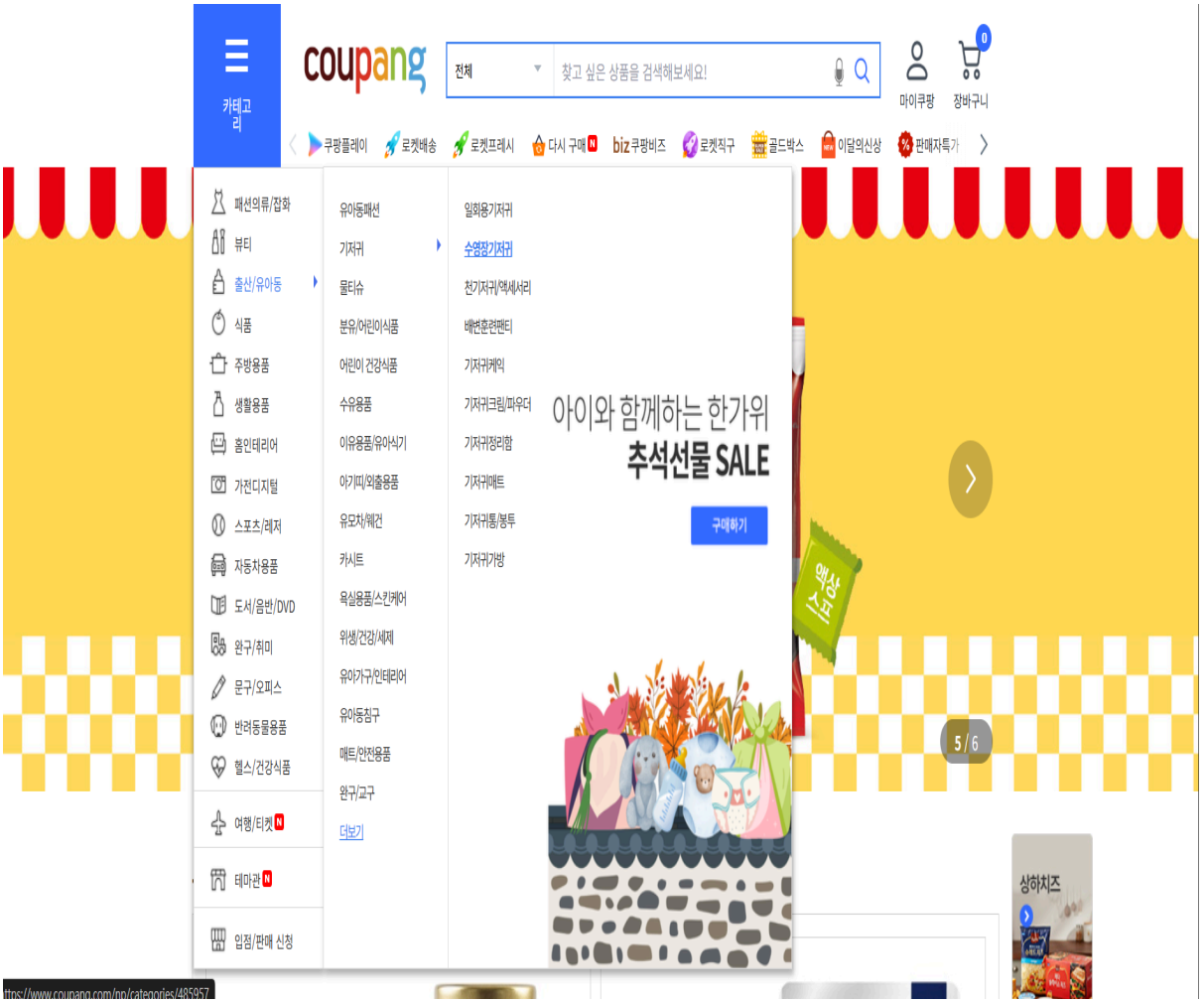
7-a. 통합 및 확장성 관리

설명 : 저희팀이 한 프로젝트는 책 관련 데이터베이스 **ERD**입니다. 당연히 주로 책에 관한 테이블이 있습니다.

근데 사업이나 회사를 운영하다 보면 사업이나 서비스를 확대해야 하는 경우도 있습니다. 이런 확장같은 경우에는 데이터베이스를 따로 구축해서 운영하면 되지 않느냐라고 생각할 수 있습니다.

하지만, 따로 구축해서 운영하는 것도 비용이고 확장 서비스가 잘 되리라는 보장이 없습니다. 코에 걸면 코걸이 귀에 걸면 귀걸이라는 말이 있듯이 책관련 데이터베이스 **ERD**를 설계하는게 아닌 포괄적인 제품 데이터베이스 **ERD**를 설계하는 관점으로 한번 만들어 보았습니다. 우선 통합적으로 관리하되 분리할 필요가 발생시 따로 분리해서 운영하는게 좋지 않을까 생각이 들었습니다. 이러한 상황을 가정하여 한번 만들어 보았습니다.

7-b. 분류 및 코드 관리



설명 : 분류는 위와 같이 쿠팡을 참조하였습니다. 분류 및 분리는 기본적으로 완벽하게 할 수는 없다 생각을 합니다. 계속해서 다른 형태의 신제품같은게 나올 여지가 있기 때문입니다.

7-c. ERD 구현 및 설명

ERD 설명 : 주로 주문, 상품, 서비스등에 대한 **ERD**로 구성하였습니다.

ERD 주소 : **https://www.erdcloud.com/d/LCNq3MhjRTXCgBW9R**

- 상태 및 요청등 : 코드성 테이블입니다. 특정 여러 테이블에 대한 요청정보나 상태정보를 저장하기 위해 만들었습니다.
  1개의 테이블로도 할 수 있으나 **1**개 테이블로 관리하면 이러한 코드들 또한 여러 개로 분리 될 수 있으므로 **2**개 정도로 분리하였습니다.
  예를 들어, 주문 실패와 같은 경우를 생각하면 주문이 실패했을 상태는 다시 주문실패/연결실패, 주문실패/잔액부족, 주문실패/재고부족 등과 같은 상황이 발생할 수 있기에 이렇게 하였습니다.
- 상품 환불및반품 : 주문상품의 상태코드를 통해 판별하도록 하였습니다. 환불 및 반품과 같은 테이블을 추가하지 않고도 상태코드등을 이용해 이러한 상태를 추적하도록 만들어 보았습니다.
- 상품카테고리 : 대분류, 중분류, 소분류 테이블로 분리하였습니다.
- 서비스 : **A/S**나 협력업체 서비스등을 고려하여 만들어 보았습니다.
- 주문상품배송 : 본인이 알기에는(잘못 알고 있을수도 있습니다.) 복합인덱스를 설정하면 첫번째, 두번째, ... 이런식으로 필터링이 된다고 알고 있기에 **PK**인덱스를 보면 주문상품배송지를 **first index**로 했는데 같은 혹은 비슷한 배송지에 대해 데이터를 먼저 필터링 해서 보여주는게 좋다고 생각을 했습니다.
- 검색 : 유저의 검색활동을 추적하기 위해 한 번 만들어보았습니다.

8-a. 데이터 생성코드

설명 : **init market.sql** 참고하시면 됩니다. 처음부터 끝까지 실행하면 됩니다.

8-b-i. 추천조회 순위 고려

추천조회 순위 고려 이유 : 배달의 민족을 보면 해당 음식점이 음식평점에 굉장히 민감해 한다는 이슈를 본 적이 있습니다. 왜냐하면 해당 음식평점이 좋냐 나쁘냐에 따라 해당 음식점의 평판과 매출이 좌지우지 되기 때문입니다. 이에 따라 해당 음식점들이 좋은 평점을 받기 위해 여러가지 애쓰는 모습이 안타까워 굳이 평점에 시달리지 않고 음식을 맛있게 만들어 파는 것에 집중할 수 있도록 해당 음식점을 노출하는 방법에 대해 한 번 고민해보았습니다.

설명 : 상품카테고리 클릭 했을 때 상품이 추천조회되는 경우를 한번 생각해보았습니다. 우선 어떻게 상품을 보여줘야 하는지에 대해서 설명을 하겠습니다.

방법**1)** 최신순 : 상품을 최신순으로 보여주게 되면 잘 팔리는 상품을 보여주기가 곤란합니다.

방법**2)** 평점 및 추천순 : 이 방법이 일반적인 방법이지만, 사용자의 추천수에 좌지우지 되는 경향이 있고 정말 좋아서 추천을 했는지 아니면 싫어서 추천을 했는지등에 대한 여부를 판단할 수가 없습니다. 게다가 블랙 컨슈머와 같이 점수에 일부러 조작을 할 수도 있기 때문입니다. 즉 상대적인 지표라 생각이 들었습니다.

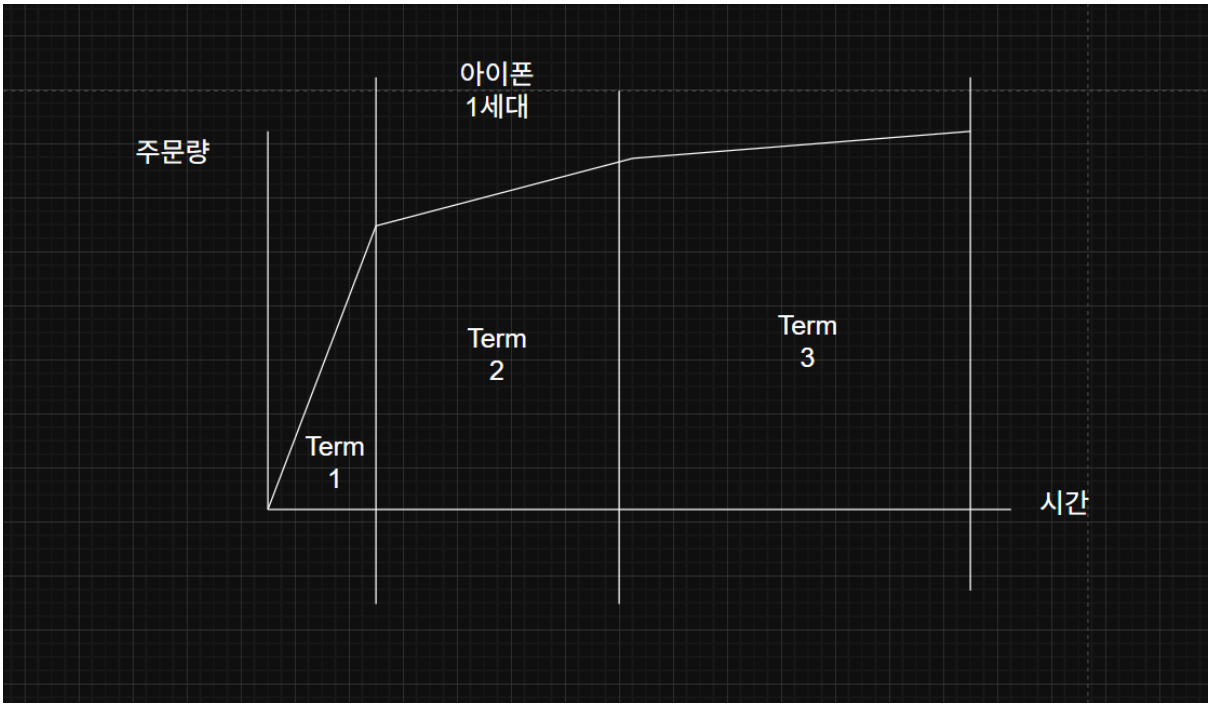
방법**3)** 판매량 및 주문량순 : 방법**2**의 상대적인 문제점을 보완하기 위해 절대적인 지표인 판매량 및 주문량순으로 표시를 하는 방법입니다. 하지만 이 방법에 대해 조금 더 자세히 알아볼 필요가 있다 생각이 들었습니다.

*그래프 모양 설명 : 어디까지나 본인이 생각한 대략적인 그래프 모양이므로 양해바랍니다.

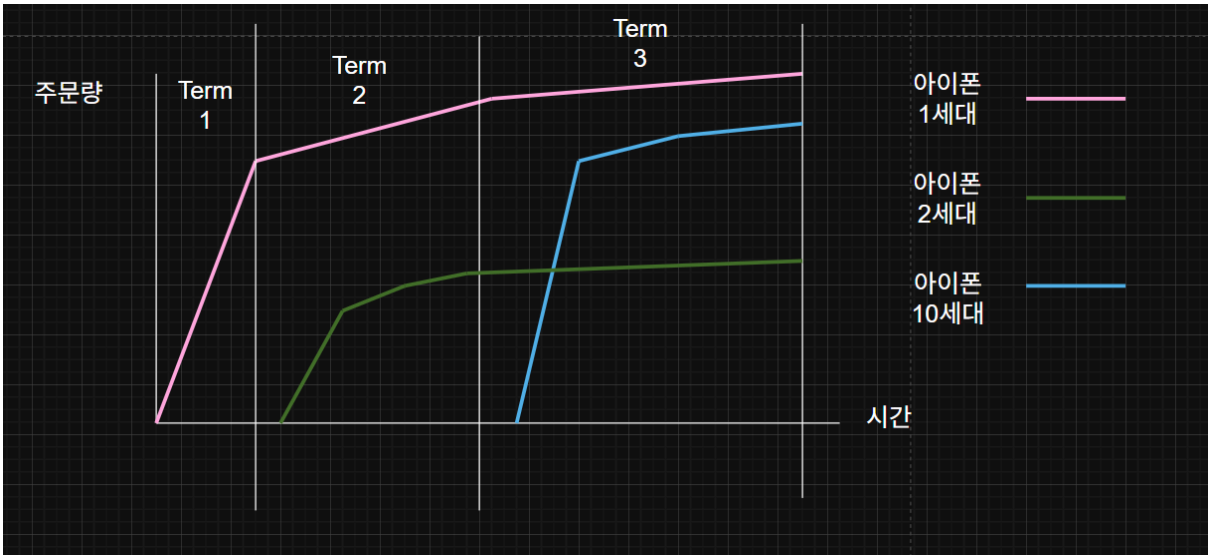
*주문량 설명 : 주문테이블에 있는 주문량을 의미합니다. 주문데이터가 삭제되지 않는다는 가정하에 설명을 하였습니다.

8-b-ii. 데이터 변화량 고려

아이폰 **1**세대의 주문량 및 판매량 **:** 아이폰 **1**세대는 최초의 상용 스마트폰이라 할 수 있습니다**.** 이 제품의 주문량이 어떨까**?**라고 생각하였을 때**,** 최초적이고 그리고 혁신적인 제품이라 갤럭시와 같은 다른 대체재 또한 없다 생각이 듭니다**.** 당연히 엄청 팔렸을 것이라 생각이 듭니다**.** 이를 그래프로 보면 다음과 같습니다**.**



설명 **:** 위의 그래프의 특징은 주문량 자체는 크지만 주문의 변화량이 시간이 지남에 따라 감소하게 됩니다**.** 그럼 이제 위의 그래프에 본인이 생각하는 아이폰 **10**세대**, 2**세대를 포함한 그래프는 다음과 같습니다**.**



설명 : 위의 그래프를 통해 알 수 있는 점이 몇가지가 있는데, 만약 단순 주문량에 대해 해당 제품을 추천조회에 올리게 되면 아이폰 **1**세대가 계속 노출이 될것 입니다. 근데 현재시간 기준으로 아이폰 **1**세대를 사용하는 사람은 거의 없습니다. 이걸 방지하기 위해 특정 유효기간**(**현재시간 **- 180**일정도부터 현재시간 까지**)**에서 특정 주문 변화량값 미만으로 떨어지면 해당 제품은 추천대상에서 제외가 되도록 하는게 좋다 판단이 들었습니다. 이유는 다음과 같습니다.

특정 주문 변화량값 설정 이유 : 실시간 주문 변화량 값 **=** 해당 제품 살 확률 이므로 이 값이 특정 주문 변화량값 보다 떨어진 판매율이 저조한 제품을 걸러내기 위해 한 번 생각해 보았습니다.

특정 유효기간 설정 이유 : 특정 유효기간 설정이 없어도 주문 변화량값은 감소하게 되어있습니다. 그런데 이 주문 변화량 값이 너무 느리게 감소하게 됩니다.

예를 들어, 아이폰 **1**세대의 **TERM 3**같은 경우에는 특정 유효기간**(**현재시간 **- 180**일정도부터 현재시간 까지**)**의 주문 변화량은 미미하지만, 모든기간**(**현재시간까지**)**의 주문 변화량은 여전히 어느정도 크기 때문입니다.

그래서 유효기간 설정이 없으면 판매율이 저조한 제품들은 장기간동안 특정 주문 변화량값을 충족하게 되므로 계속해서 보여지게 될거라 생각이 듭니다.

결론 : 이렇게 함으로써 판매율이 좋은 제품은 노출하고 판매율이 저조한 제품은 노출하지 않는 기능을 실시간으로 적용을 하는 기능을 한 번 생각해 보았습니다.

8-c. 데이터 추천조회 코드

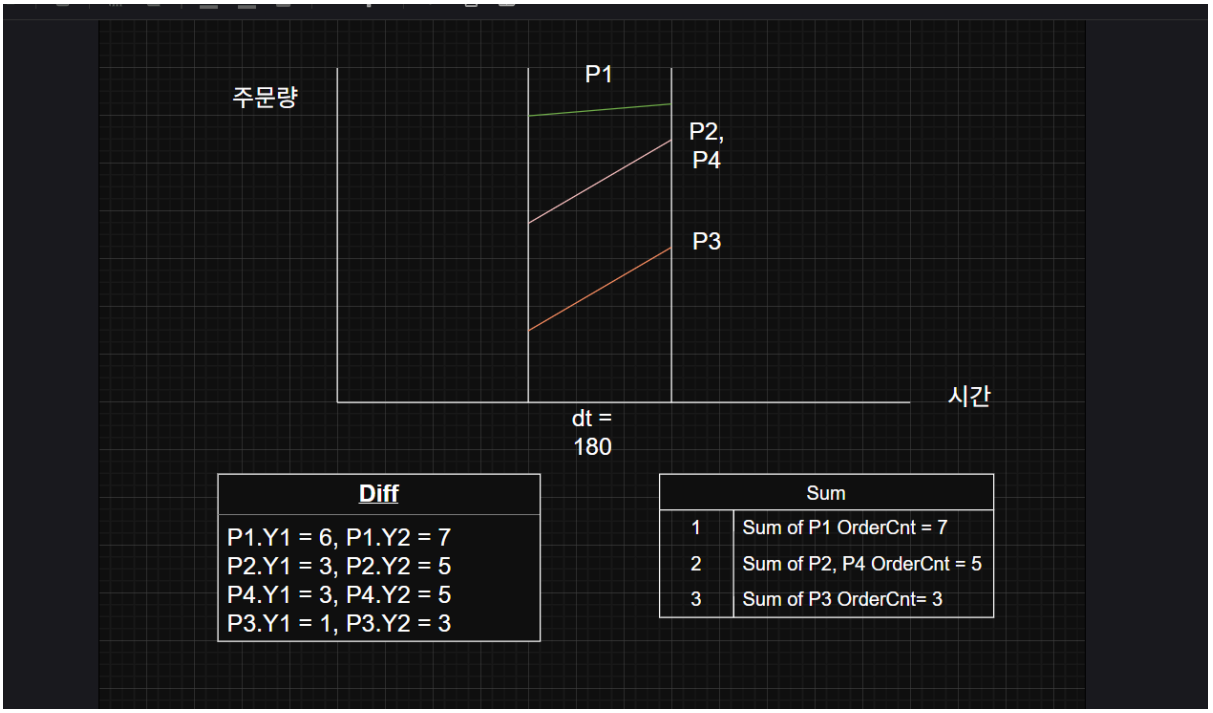
설명 : **sql for market.sql** 참고하시면 됩니다. 카테고리 테이블에 있는 기준값들은 서버에서 패러미터로 받아와서 처리하는 걸로 가정하였습니다.

버전**1** 코드는 처음본인이 생각해낸 코드이고 버전**2**코드는 버전**1**코드를 약간 개량하여 코드를 구성해보았습니다. 결과는 버전**2** 코드의 결과입니다.

추가 고려점 : 데이터 조회는 추천조회 뿐만 아니라 평점순, 판매량순, 최신순도 포함하는게 좋다고 생각됩니다. 왜냐하면 소비자는 제품에 대한 여러가지 순서를 볼 필요가 있으며 추천조회가 항상 소비자가 원하는게 아닐 수도 있기 때문입니다.

- 추천조회순위 :
  - **1**순위 : 델타값보다 큰가
  - **2**순위 : 판매량 순
  - **3**순위 : 주문 변화량값 순
  - **4**순위 : 평점 순
  - **5**순위 : 평점 횟수 순
  - **6**순위 : 상품 열람수 순

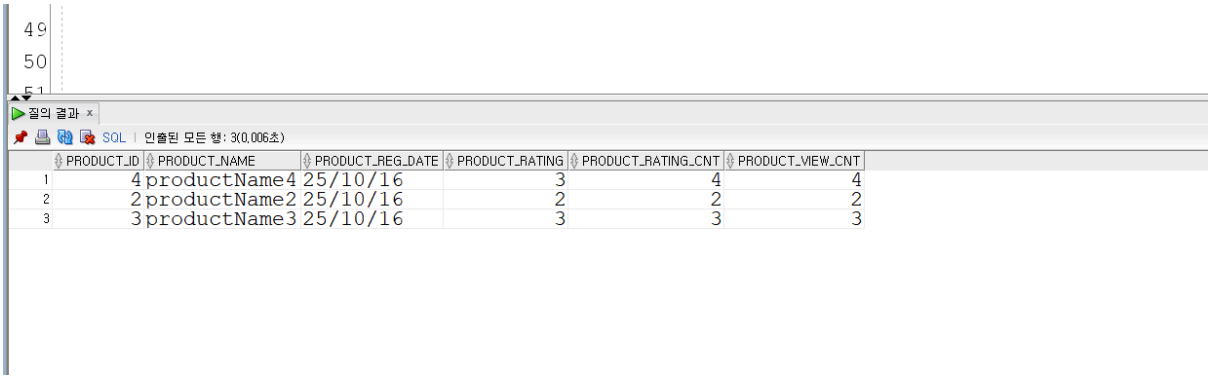
참고 그래프 :



참고그래프 설명 : 기준 주문 변화량값은 **0.01**로 유효기간은 **180**일로 정해보았습니다. **SUM**표는 각각의 상품**(P1, P2, P3, P4)**의 현재 주문량 값입니다. 이값이 **DIFF**표의 **Y2**값이며 **Y1**값은 **180**일 이전의 주문량 값입니다.

코드 및 결과 : **P1**은 기준 주문 변화량값이 기준 주문량값에 충족 안돼서 제외되었습니다. **P3**는 주문량 저조해서 뒤로 밀려나게 되었습니다. **P2, P4** 중 위의 순위 고려하여 표시를 하면 다음과 같습니다.

```sql
SELECT p.product_id product_id, --버전2
    MAX(p.product_name) product_name,
    MAX(p.product_reg_date) product_reg_date,
    MAX(p.product_rating) product_rating,
    MAX(p.product_rating_cnt) product_rating_cnt,
    MAX(p.product_view_cnt) product_view_cnt
    FROM Orders o
    INNER JOIN OrderProduct op ON (o.order_id = op.order_id)
    INNER JOIN Product p ON (op.product_id = p.product_id)
    WHERE p.prod_sl_ctg_id = 1
GROUP BY p.product_id
HAVING (COUNT(p.product_id) - (SELECT COUNT(*)
        FROM Orders o1
        INNER JOIN OrderProduct op1 ON (o1.order_id = op1.order_id)
        WHERE TO_DATE(o1.order_date, 'YYYY-MM-DD') <= TO_DATE(SYSDATE - 180, 'YYYY-MM-DD')
        AND op1.product_id = p.product_id)) / 180 > 0.01
ORDER BY COUNT(p.product_id) DESC,
(COUNT(p.product_id) - (SELECT COUNT(*)
FROM Orders o1
INNER JOIN OrderProduct op1 ON (o1.order_id = op1.order_id)
WHERE TO_DATE(o1.order_date, 'YYYY-MM-DD') <= TO_DATE(SYSDATE - 180, 'YYYY-MM-DD')
AND op1.product_id = p.product_id)) / 180 DESC,
MAX(p.product_rating) DESC, MAX(p.product_rating_cnt) DESC, MAX(p.product_view_cnt) DESC;
```



9. 프로젝트 느낀점

설명 : 초기 **1**차 프로젝트**(**커뮤니티 프로젝트**)**보다는 보다 원활히 진행할 수 있었습니다**.** 초기 **1**차 프로젝트는 제가 아무래도 코딩을 해본 경험이 있어 제 역할이 많았지만 팀원분들이 **1**차 프로젝트를 진행하면서 동시에 실력이 많이 올라 저에게 많은 도움을 주어서 고마웠습니다**.**