

목차

1. ERD 고려점

- a. 통합 및 확장성 관리
- b. 분류 및 코드 관리
- c. 구현 및 설명

2. SQL 고려점

- a. 데이터 생성 코드
- b. 상품 조회 고려
 - i. 조회 순위 고려
 - ii. 데이터 변화량(델타) 고려
 - iii. 델타의 변화량 고려
 - iv. 부분 데이터 추출 및 고려
- c. 데이터 조회 코드

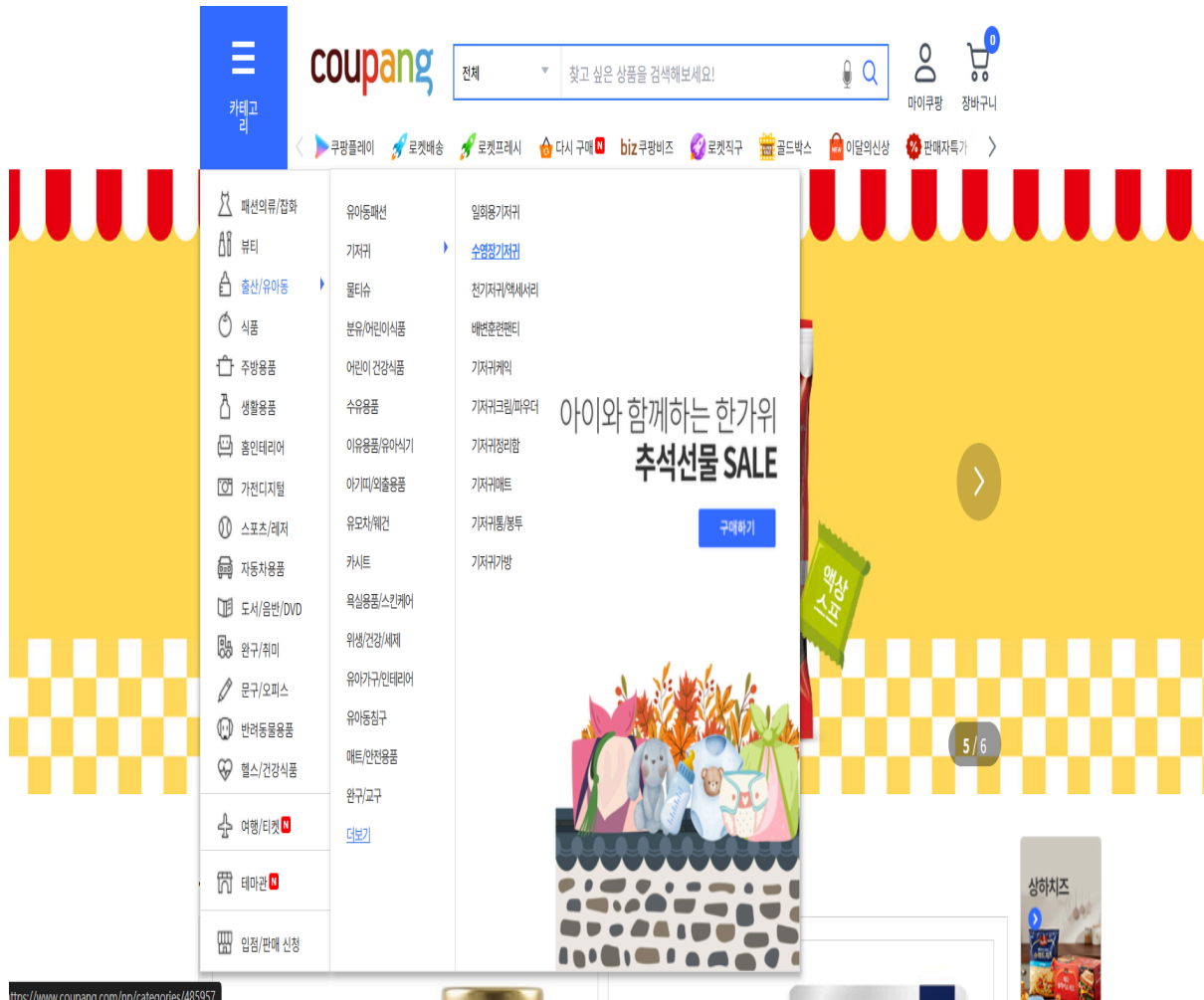
1-a. 통합 및 확장성 관리

설명 : 본인이 한 프로젝트는 책 관련 데이터베이스임 당연히 책에 관한 테이블만 들어있음.

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

하지만, 따로 구축해서 운영하는 것도 비용이고 확장 서비스가 잘 되리라는 보장이 없음. 코에 걸면 코걸이 귀에 걸면 귀걸이라는 말이 있음. 그래서 책관련 데이터베이스를 설계하는게 아닌 포괄적인 제품 데이터베이스를 설계하는 관점으로 함. 우선 통합적으로 관리하되 잘되면 분리해서 따로 운영하는게 좋지 않을까 판단.

1-b. 분류 및 코드 관리



설명 : 분류는 위와 같이 쿠팡을 참조함. 분류 및 분리는 기본적으로 완벽하게 할 수 없음. 계속해서 다른 형태의 신제품같은게 나올 여지가 있기 때문. 테이블 코드를 통해 분류/분리할 예정임.

1-c. ERD 구현 및 설명

ERD 설명 : 주로 주문과 상품에 대한 ERD로 구성함. 나머지 세세한 기능(고객센터, 부서, 약관, 관리자등)들은 생략함.

ERD 주소 : <https://www.erdcloud.com/d/LCNq3MhjRTXCgBW9R>

- **상태 및 요청등** : 코드성 테이블임. 해당 테이블에 대한 요청정보나 상태정보를 저장하기 위해 만듦. 1개의 테이블로도 할 수 있으나 1개 테이블로 관리하면 사람입장에서 보기 어렵고 이러한 코드들 또한 여러 개로 분류 및 분리 될 수 있으므로 타겟 테이블의 **FK**로

설정함. 여러 테이블에도 써먹을려고 이렇게 함. 단점은 설계 잘못하면 꼬이는 단점이 있음.

- **상품카테고리** : 위의 쿠팡 사진 참고.
- **상품 환불및반품** : 주문상품의 상태코드를 통해 판별. 테이블추가 **X**.
- **서비스** : **A/S**나 협력업체 서비스등 고려.
- **주문상품배송** : 인덱스를 보면 주문상품배송지를 **first index**로 했는데 주문상품배송지로 먼저 묶어서 하는게 좋다고 판단. 주문이나 상품아이디를 기준으로 배송을 하면 배송이 서울에 갈 수도 있고 부산에도 갈 수 있음. 이러면 비효율적이기 때문에 같은 혹은 비슷한 배송지역에 대한 배송을 하는게 효율적이라 판단. 즉 지역묶음배달을 하기위함임. 배송 관리자를 위한 코드로 관리하는게 좋다 생각.

데이터 많은 곳 : 위의 **ERD** 에서 데이터가 어느 테이블에 가장 많이 쌓이고 가장 많이 조회되는가는 본인이 생각하기엔 유저, 주문, 주문상품, 검색등일 것이라 판단됨.

2-a. 데이터 생성코드

설명 : **init market.sql** 참고바람. 처음부터 끝까지 실행하면됨.

2-b-i. 조회 순위 고려

설명 : 상품카테고리 클릭 했을 때 상품이 조회되는 경우를 소개함. 우선 어떻게 상품을 보여줘야 하는지에 대해서 설명함.

방법1) 최신순 : 상품을 최신순으로 보여주게 되면 잘 팔리는 상품을 보여주는게 곤란하다.

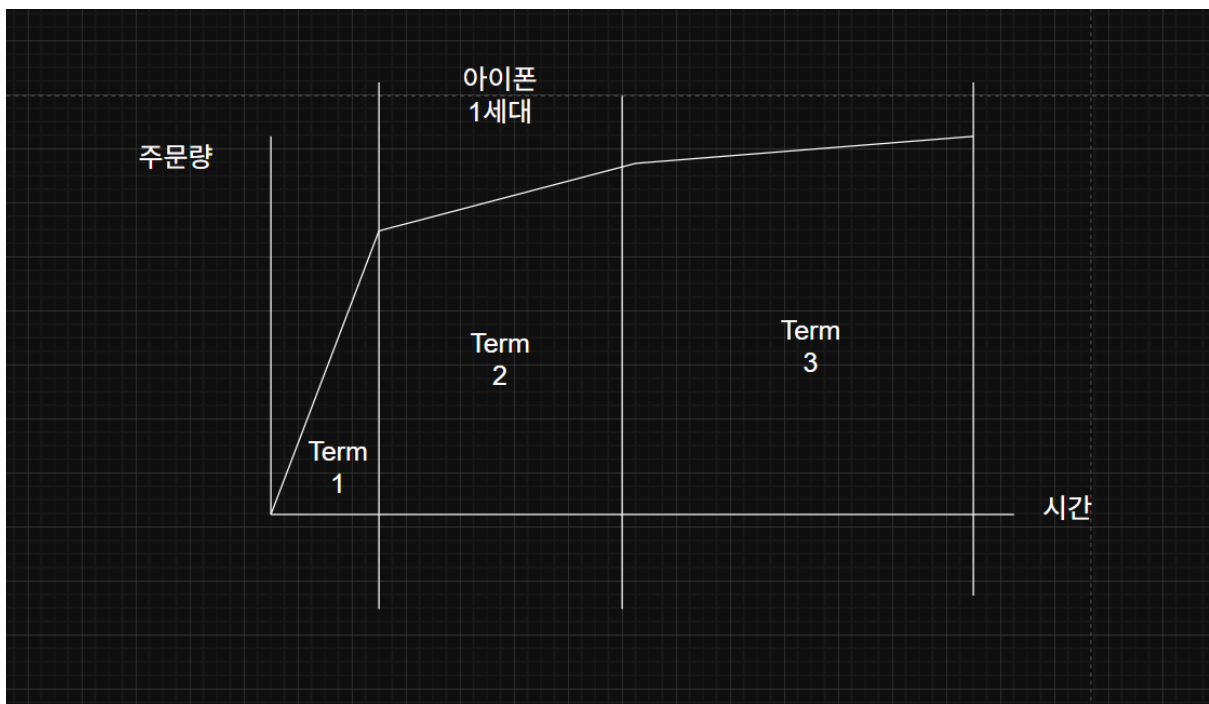
방법2) 평점 및 추천순 : 이 방법이 일반적인 방법이지만, 사용자의 추천수에 좌지우지 되는 경향이 있고 정말 좋아서 추천을 했는지 아니면 싫어서 추천을 했는지등에 대한 여부를 판단할 수가 없다. 게다가 블랙 컨슈머와 같이 점수에 일부러 조작을 할 수도 있음. 즉 상대적인 지표임. 그리고 본인에게 있어 유저 데이터 요청은 신뢰하지 않는 편이다.

방법3) 판매량 및 주문량순 : 방법2의 상대적인 문제점을 보완하기 위해 절대적인 지표인 판매량 및 주문량순으로 표시를 하는 방법임. 하지만 이 방법에 대해 조금 더 자세히 알아볼 필요가 있음.

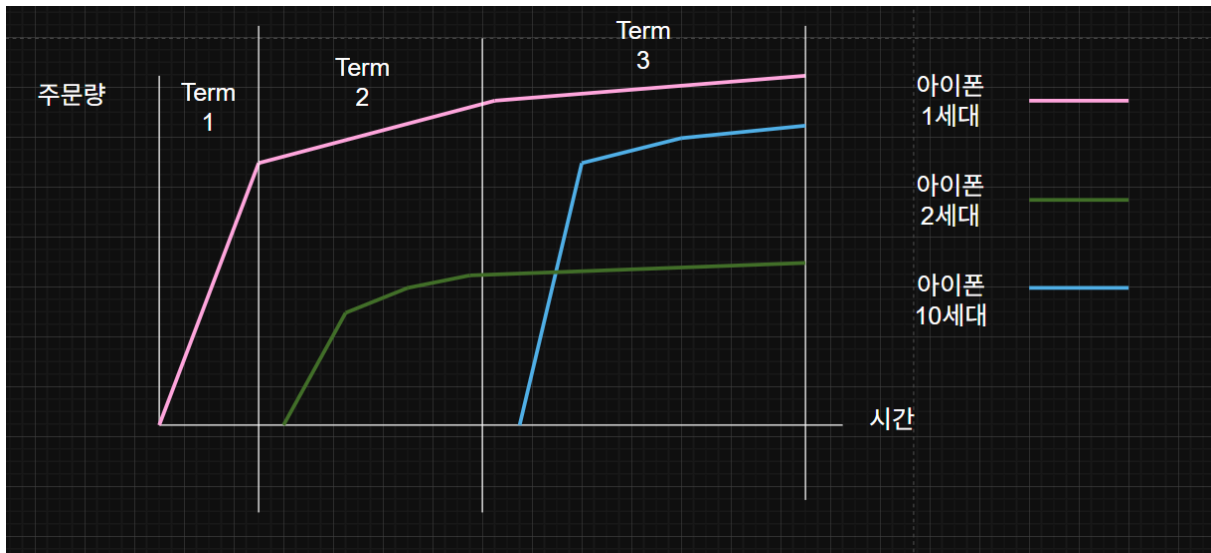
***그래프 모양 설명** : 어디까지나 본인이 생각한 대략적인 그래프 모양이므로 양해바랍니다.

2-b-ii. 데이터 변화량 고려

아이폰 1세대의 주문량 및 판매량 : 아이폰 1세대는 최초의 상용 스마트폰이라 할 수있다. 이 제품의 주문량이 어떨까? 최초적이고 그리고 혁신적인 제품이라 갤럭시와 같은 다른 대체재 또한 없다. 당연히 엄청 팔릴 것이다. 이를 그래프로 보면 다음과 같다.



설명 : 위의 그래프의 특징은 주문량 자체는 크지만 주문량의 변화량(이하 델타라 함)이 시간이 지남에 따라 감소하게 된다. 그럼 이제 위의 그래프에 아이폰 10세대, 2세대를 포함한 그래프는 다음과 같다.



설명 : 위의 그래프를 통해 알 수 있는 점이 몇가지가 있는데, 만약 단순 주문량에 대해 해당 제품을 추천조회에 올리게 되면 아이폰 **1세대**가 계속 노출이 될것이다. 근데 아이폰 **1세대**를 사용하는 사람이 있는가? 거의 없다. 이걸 방지하기 위해 특정 유효기간에서 특정 델타값 미만으로 떨어지면 해당 제품은 추천대상에서 제외가 된다. 이유는 다음과 같다.

특정 델타값 설정 이유 : 델타값 = 주문 변화량값 = 해당제품 기호도 = 해당 제품 살 확률 이므로 아이폰 **1세대**와 같은 구시대적인 제품을 걸러내기 위해 필요.

특정 유효기간 설정 이유 : 특정 유효기간 설정이 없어도 델타값은 감소하게 되어있다. 그런데 이 델타값이 너무 느리게 감소한다. 그래서 유효기간 설정이 없으면 아이폰 **1세대**와 같은 제품들은 장기간동안 델타값을 충족하게 되므로 계속해서 보여지게 된다.

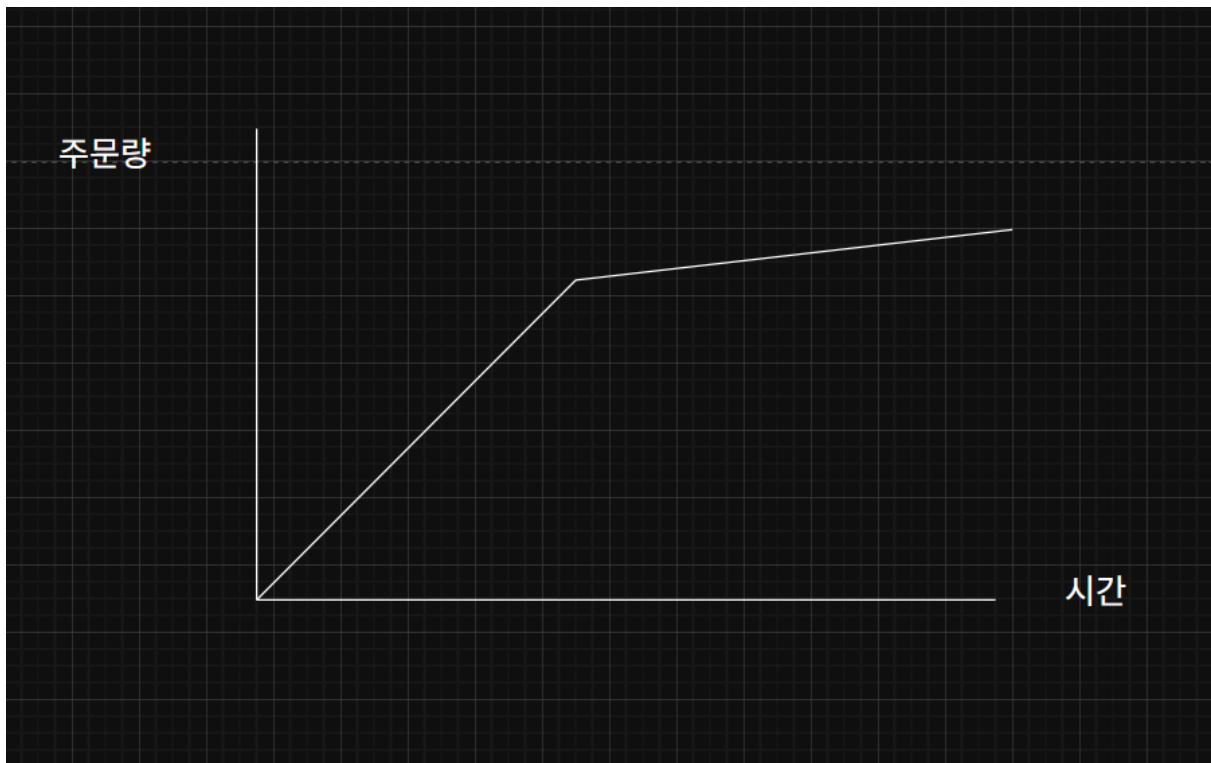
2-b-iii. 델타의 변화량 고려

설명 : 델타의 변화량에 대해 다음의 **2**가지 상황에 대해 설명을 함.

상황 1 : 어떤 상품이 갑자기 단종되거나 이러한 경우. 즉, 희귀성이 올라가는 경우.



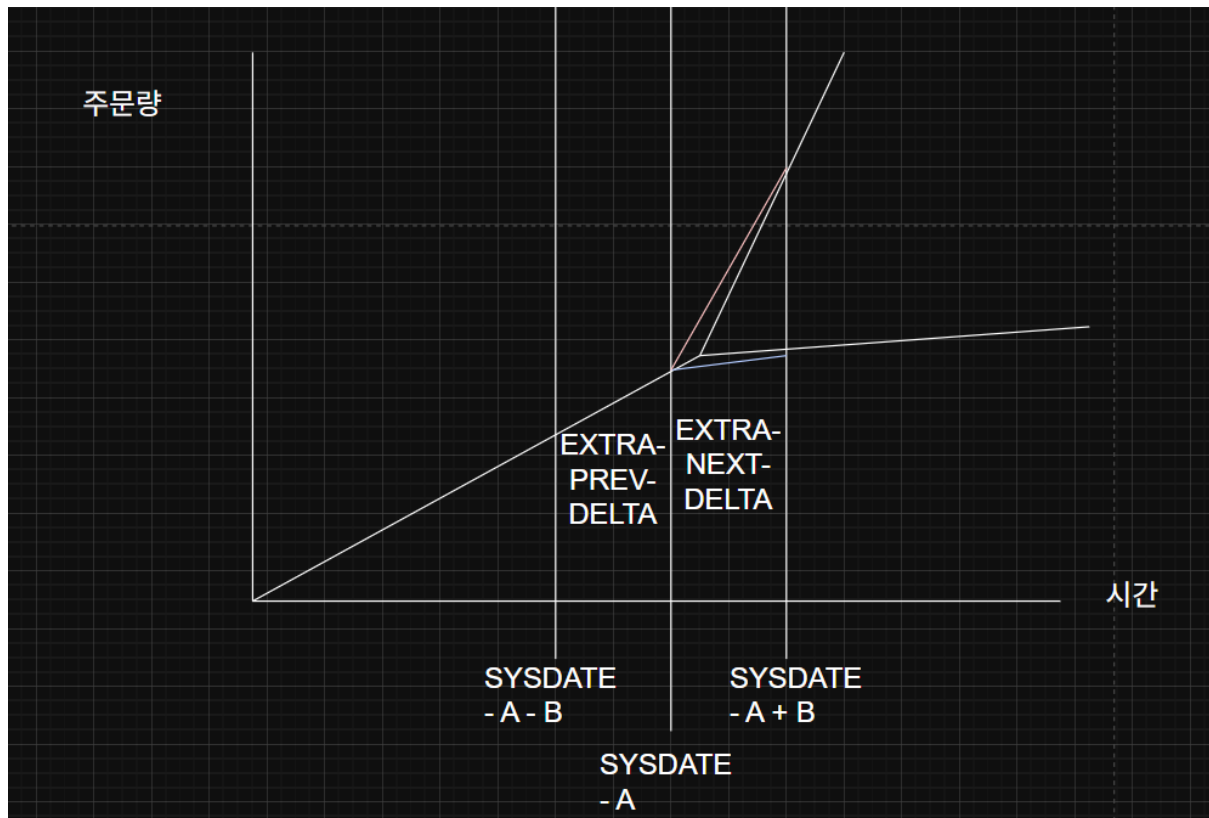
상황 2 : 어떤 상품에 대해 갑자기 문제가 생기는 경우. 배터리 이슈등.



이러한 상황들 고려점 : 상황**1**에 대해선 해당제품회사로 전화해 물량상태등을 체크하는것이 좋을것이고 상황**2**에 대해선 해당 회사로

전화를해 미리 불량상태등을 점검해 제품을 검수하는게 좋다 판단.
이상한 제품을 보내면 소비자들은 싫어할 수 있음.

구현 방법 :



설명 : **SYSDATE - A**를 기준으로 **EXTRA-NEXT-DELTA**와 **EXTRA-PREV-DELTA**를 구한 뒤, 이 차이값을 구해 양수면 급등, 음수면 급강. 그리고 기준 델타값도 설정해야함(얼마만큼 올랐냐 내렸냐). **A**와 **B**의 값은 어느 정도 작게 해야할 필요가 있다 판단됨. 또한, 급등 및 급강을 확인해야 하므로 **A**와 **B**의 값은 작게 설정할 필요가 있음. 크게 설정하면 급등 및 급강의 크기가 지속적으로 감소하고 이러한 지속성은 이 상황에는 어울리지 않음.

2-b-iv. 부분 데이터 추출 및 고려

설명 : 이 델타 값들을 세세히 쪼개면 더욱 정교한 분석이 가능함. 정적으로도 가능하지만, 동적으로도 하는 것도 한번 생각을 해봄.

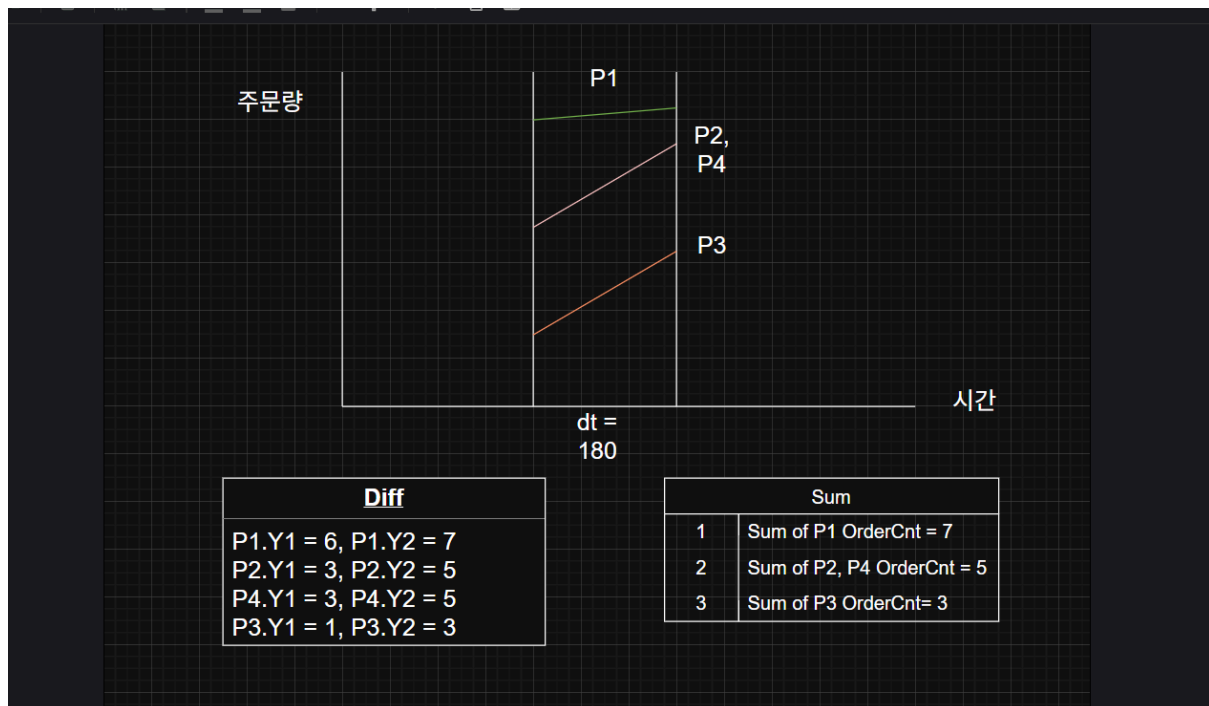
이러한 과정을 통해 어떤 인사이트를 얻으면 미래의 어떤 경향도 대략적으로 유추할 수 있다 판단. 하지만 전문적인 지식과 기술이 필요할 것 같은 단점이 있을거라 판단됨. 해당 데이터를 저장하기위한 공간또한 많이 필요할 거라 판단됨.

2-c. 데이터 조회 코드

설명 : **sql for market.sql** 참고. 카테고리 기준값은 서버에서 패러미터로 받아와서 처리하는 걸로 함.

- **조회순위** :
 - **1순위** : 델타값보다 큰가
 - **2순위** : 판매량 순
 - **3순위** : 델타값 순
 - **4순위** : 평점 순
 - **5순위** : 평점 횟수 순
 - **6순위** : 상품 열람수 순

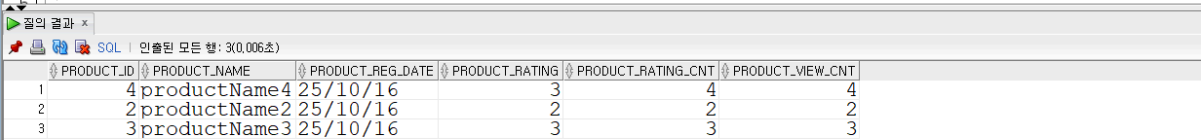
참고 그래프 :



설명 : **P1**은 델타 충족 안돼서 제외됨. **P3**는 주문량 저조해서 뒤로 밀려남.

P2, P4 중 위의 순위 고려하여 표시됨.

결과 :



| | PRODUCT_ID | PRODUCT_NAME | PRODUCT_REG_DATE | PRODUCT_RATING | PRODUCT_RATING_CNT | PRODUCT_VIEW_CNT |
|---|------------|--------------|------------------|----------------|--------------------|------------------|
| 1 | 4 | productName4 | 25/10/16 | 3 | 4 | 4 |
| 2 | 2 | productName2 | 25/10/16 | 2 | 2 | 2 |
| 3 | 3 | productName3 | 25/10/16 | 3 | 3 | 3 |

코드 고려점 : 주문, 주문량, 상품에 대한 조인->**GROUP BY** 이므로 연산부하가 심할거라 예상이됨. SQL 튜닝 필요하다 판단.