BigQuery Partitioning & Clustering 설계 (Final)

1, 목적 **(Why)**

본 프로젝트는 대규모 Raw 테이블(events, sessions, orders)을 기반으로 Data Mart(DM) 및 분석 SQL을 반복 실행하기 때문에 Partitioning & Clustering 을 설계하였습니다.
따라서 BigQuery 비용/성능 최적화를 위해 아래 원칙으로 **Partitioning & Clustering**을 적용용.

- **Partitioning**: 시간 기반 필터(날짜 범위)로 스캔 범위를 줄이기 위한 1차 최적화
- **Clustering**: 조인/집계에서 반복적으로 사용되는 키 기준으로 데이터 locality를 높여 2차 최적화
- 목표는 가능한 많은 컬럼을 넣는 것이 아니라, 반복 쿼리 패턴에 가장 자주 등장하는 핵심 키로 최소화하는 것

2, 쿼리 패턴 가정 **(Query Patterns)**

실제 분석/DM 생성에서 자주 반복되는 패턴은 아래와 같습니다.

- user_id 기준 유저 단위 집계 (activation/ltv/retention/consistency)
- session_id 기준 세션 단위 funnel 분석 (특히 DM_funnel_session)
- event_type 필터링/집계 (view/click/add_to_cart/checkout/purchase)
- 대부분 쿼리는 날짜 조건(기간 필터)이 존재하며, full scan을 최대한 피하기 위해 설정정

3, Raw 테이블 최종 설계 **(Final)**

2.1 events

- **Partition**: DATE(event_ts)
- **Cluster**: user_id, session_id (조인키 priority), event_type (보조)

선정 이유

- events는 가장 크고 스캔 비용이 큰 테이블이므로 시간 필터 + (user/session/event_type) 조합이 효율적
- product_id는 특정 분석에서만 쓰이므로 기본 clustering에서 제외 (옵션으로만 고려)
  - v1 scope 에서 **promo** 관련 분석을 제외하기로 결정해서**, product_id** 는 제외

DDL

CREATE TABLE ‘eternal-argon-479503-e8.ecommerce.events_p’

PARTITION BY DATE(event_ts)
CLUSTER BY user_id, session_id, event_type AS
SELECT * FROM ‘eternal-argon-479503-e8.ecommerce.events’;

Before Partitioning/Clustering

After Partitioning/Clustering

2.2 sessions

- **Partition**: DATE(session_start_ts)
- **Cluster**: user_id, session_id

선정 이유

- 세션 기반 분석은 user_id/session_id 조인이 핵심

DDL

CREATE TABLE ‘eternal-argon-479503-e8.ecommerce.sessions_p’
PARTITION BY DATE(session_start_ts)
CLUSTER BY user_id, session_id AS
SELECT * FROM ‘eternal-argon-479503-e8.ecommerce.sessions’;

Before Partitioning/Clustering

After Partitioning/Clustering

2.3 orders

- **Partition**: DATE(order_ts)
- **Cluster**: user_id

선정 이유

- orders는 LTV/재구매/구매 여부 등 대부분이 유저 단위 집계로 사용
- order_id는 고유값(unique)이므로 clustering 효율이 제한적 → 제외

DDL

CREATE TABLE ‘eternal-argon-479503-e8.ecommerce.orders_p’
PARTITION BY DATE(order_ts)
CLUSTER BY user_id AS
SELECT * FROM ‘eternal-argon-479503-e8.ecommerce.orders’;

Before Partitioning/Clustering

After Partitioning/Clustering

4, 테이블 스왑(rename) 전략

기존 테이블을 보관하면서 새 파티션/클러스터 테이블로 안전하게 교체하였습니다.

ALTER TABLE ‘eternal-argon-479503-e8.ecommerce.events’ RENAME TO ‘events_old’;
ALTER TABLE ‘eternal-argon-479503-e8.ecommerce.events_p’ RENAME TO `events`;

ALTER TABLE ‘eternal-argon-479503-e8.ecommerce.sessions’ RENAME TO ‘sessions_old’;
ALTER TABLE ‘eternal-argon-479503-e8.ecommerce.sessions_p’ RENAME TO ‘sessions’;

ALTER TABLE ‘eternal-argon-479503-e8.ecommerce.orders’ RENAME TO ‘orders_old’;
ALTER TABLE ‘eternal-argon-479503-e8.ecommerce.orders_p’ RENAME TO ‘orders’;

5, 설계 결정 요약 (Design Decisions)

Clustering 키 최소화 원칙

초기 설계에서는 보조 분석까지 고려해 클러스터 키를 넓게 잡을 수 있으나,
실제 반복 실행되는 핵심 쿼리 패턴(user/session 중심) 대비 과한 키는 효율을 떨어뜨릴 수 있다고 생각했습니다, 따라서 최종적으로 핵심 키로 단순화 했습니다.

결과값 영향 없음

Partitioning/Clustering은 저장 구조 최적화이며, 쿼리 결과(집계/전환율/LTV/retention)는 변하지 않습니다다. 즉, 기존 분석 결과 스크린샷/CSV를 “다시 찍을 필요”는 없고, 성능/비용 최적화 관점에서만 의미가 있습니다다.

Query guideline

차후 Data mart 생성 때 생각해야할 필터 패턴: 파티션 컬럼을 그대로 사용하거나 timestamp 범위를 직접 사용해서 Partition pruning 이 깨지지 않게 주의의

Sanity check code

```
SELECT ‘events’ ,
  (SELECT COUNT(*) FROM `eternal-argon-479503-e8.ecommerce.events`) AS old_cnt,
  (SELECT COUNT(*) FROM `eternal-argon-479503-e8.ecommerce.events_p`) AS new_cnt
UNION ALL
SELECT 'sessions',
  (SELECT COUNT(*) FROM `eternal-argon-479503-e8.ecommerce.sessions`),
  (SELECT COUNT(*) FROM `eternal-argon-479503-e8.ecommerce.sessions_p`)
UNION ALL
SELECT 'orders',
  (SELECT COUNT(*) FROM `eternal-argon-479503-e8.ecommerce.orders`),
  (SELECT COUNT(*) FROM `eternal-argon-479503-e8.ecommerce.orders_p`);
```