İÇİNDEKİLER

1. Mimari Yapı

1.1. Açıklama

Bu e-ticaret veri ambarı projesinde, mimariyi Google Cloud Platform (GCP) servislerini kullanarak Medallion (Bronze, Silver, Gold) mimarisine ve Star Schema modelleme tekniğine dayandırarak kurgulayacağım. Veri işleme yükünün tamamını, dbt (Data Build Tool) kullanarak BigQuery üzerinde yöneteceğim.

Bu yaklaşımı seçmemin nedeni, GA4 verisinin BigQuery'ye kodsuz ve otomatik aktabilmesidir. Bu veriyi BigQuery dışına çıkarmadan, doğrudan BigQuery'nin kendi işlem gücünü kullanarak (ELT) işlemek en maliyet etkin yöntemdir. dbt'yi bu süreci yönetmek için kullanacağım çünkü SQL tabanlı bu dönüşümleri test edilebilir, versiyonlanabilir ve CI/CD süreçlerine uygun bir mühendislik disipliniyle geliştirmemizi sağlar. Medallion mimarisi veri kalitesini ve geriye dönük izlenebilirliği garanti altına alırken, son katmandaki Star Schema ise analistlerin ve BI araçlarının veriye en hızlı ve en kolay şekilde erişmesini sağlayacaktır.

1.2. Medallion Mimari

Layer 1: Bronze (Ham Veri Katmanı)

Amaç: Kaynak (GA4) verisini, üzerinde hiçbir değişiklik yapmadan, ham haliyle depolamak.

Teknoloji: BigQuery (Native GA4 Export).

Süreç:

Veri Alımı: GA4'ün "BigQuery Export" özelliği kullanılarak, verinin günlük (batch) olarak BigQuery projemize otomatik akmasını sağlayacağım. Bu, "kod gerektirmeyen" (no-code) ve en verimli ingestiyon yöntemidir.

Depolama: Veri, BigQuery'de events_YYYYMMDD formatında, günlük partition'lı (bölümlenmiş) tablolarda saklanacaktır.

Layer 2: Silver (Temizlenmiş ve Yapılandırılmış Katman)

Amaç: Bronze katmanındaki ham ve karmaşık veriyi temizlemek, yapılandırmak ve analize daha uygun hale getirmek. JSON'ları açmak (un-nesting), veri tiplerini düzeltmek ve temel iş kurallarını uygulamak bu katmanın görevidir.

Teknoloji: dbt & BigQuery.

Süreç:

Dönüşüm: dbt kullanarak BigQuery üzerinde SQL tabanlı dönüşüm modelleri geliştireceğim. Asıl işlem yükünü BigQuery'nin işlem gücü üstlenecektir. dbt bu süreci yönetir ve SQL'leri BigQuery'ye "push-down" eder.

Un-nesting: event_params, items gibi JSON yapılarını UNNEST fonksiyonu ile açarak sütun bazlı (columnar) bir yapıya getireceğim.

Temizlik: Veri tiplerini standartlaştıracağım (örn: event_timestamp mikrosaniyeden timestamp formatına) ve user_pseudo_id veya session_id olmayan verileri ayıklayacağım.

Depolama: Bu katmanda stg_events, stg_sessions gibi "staging" tabloları oluşturulur.

Layer 3: Gold (İş Odaklı Veri Katmanı)

Amaç: Son kullanıcıların (Analistler, Data Scientist'ler, BI araçları) kolayca sorgu atabileceği, yüksek performanslı ve iş metriklerini içeren veri mart'ları oluşturmak. Bu katmanı, Star Schema yapısında modelleyeceğim.

Teknoloji: dbt & BigQuery.

Süreç:

Modelleme: Silver katmanındaki tablolardan beslenerek Fact ve Dimension tablolarını dbt modelleri olarak oluşturacağım.

Fact Tabloları: fct_ecommerce_transactions, fct_item_sales gibi işlem bazlı metrikleri içerir.

Dimension Tabloları: dim_user, dim_device, dim_geo gibi olayın bağlamını sağlar.

Depolama: Bu tablolar BigQuery'de, analiz için cluster edilmiş son tablolar olarak saklanır.

Orkestrasyon: Cloud Composer (Managed Airflow). Günlük dbt modellerinin (Bronze -> Silver -> Gold) sıralı ve bağımlılıkları yönetilerek schedule edilmesini sağlar. DbtRunOperator veya BashOperator (dbt run komutu için) kullanılacaktır.

1.3. Star Schema Tasarımı

Bu bölüm, transformasyon aracından bağımsız olarak hedef veri modelini tanımlar ve bir önceki cevabımla aynıdır, çünkü hedefimiz (Star Schema) değişmemiştir.

Fact Tabloları

fct_ecommerce_transactions

- **Grain:** Her satır bir ecommerce.transaction_id.
- **Sütunlar:** transaction_id (PK), date_key (FK), user_key (FK), session_key (FK), device_key (FK), geo_key (FK), purchase_revenue_in_usd (Metrik), total_item_quantity (Metrik).

fct_item_sales

- **Grain:** Her satır, bir siparişteki bir item_id.
- **Sütunlar:** transaction_id (FK), item_key (FK), date_key (FK), user_key (FK), item_quantity (Metrik), item_price (Metrik), item_revenue_in_usd (Metrik).

fct_session_summary

- **Grain:** Her satır bir session_id.
- **Sütunlar:** session_key (PK/FK), date_key (FK), user_key (FK), device_key (FK), geo_key (FK), traffic_key (FK), session_duration_sec (Metrik), pageview_count (Metrik), is_conversion (Metrik).

1.4 Dimension Tabloları

dim_date

- **Sütunlar:** date_key (YYYYMMDD), date, day_of_week, month, year, is_weekend

dim_user

- **Sütunlar:** user_key (Surrogate Key), user_pseudo_id, user_id, first_seen_timestamp, last_seen_timestamp

dim_session

- **Sütunlar:** session_key (Surrogate Key), session_id, session_number

dim_traffic_source

- **Sütunlar:** traffic_key (Surrogate Key), traffic_source_medium, traffic_source_source

dim_device

- **Sütunlar:** device_key (Surrogate Key), device_category, mobile_brand_name, operating_system

dim_geo

- **Sütunlar:** geo_key (Surrogate Key), country, city
- **dim_item**
- **Sütunlar:** item_key (Surrogate Key), item_id, item_name, item_category

2. Maliyet Optimizasyonu

2.1. Depolama (Storage) Maliyet Optimizasyonu

- **Partitioning :** Tüm katmanlardaki (Bronze, Silver, Gold) zamana bağlı tablolar, özellikle de event_timestamp sütununa göre partitioned olarak oluşturulacak. GA4 verisi bir zaman serisi verisidir ve sorguların çoğu "son 7 gün", "dün" gibi zaman filtreleri kullanır. Bu yöntemle BigQuery, bir sorgu çalıştırdığında 1 yıllık tüm veriyi taramak yerine *sadece* ilgili tarih aralığındaki partition'ları tarar. Bu, taradığı veri miktarını ve dolayısıyla sorgu maliyetini büyük ölçüde düşürür.
- **Clustering :** Partitioning genelde tarih üzerinden yapılır. Buna ek olarak, Gold katmanındaki tabloları (fct_transactions gibi) user_key (veya user_pseudo_id) ve session_key gibi sık filtrelenen sütunlara göre cluster edilecek. Bu, "belirli bir kullanıcının son 30 gündeki hareketleri" gibi sorgularda, BigQuery'nin tarih partition'ı *içinde* tüm veriyi okumak yerine, verinin fiziksel olarak sıralandığı ilgili user_key bloklarına gitmesini sağlar. Bu da taradığı veriyi azaltarak sorguları hızlandırır ve ucuzlatır.
- **TTL - Time To Live :** Her veriyi sonsuza kadar saklamak maliyetlidir. Bronze (Ham Veri) katmanı için 180 gün (veya iş ihtiyacına göre 90 gün) gibi bir Partition Expiration (TTL) ayarlanacak. Benzer şekilde, sadece Gold'u besleyen Silver (Ara Katman) için 30-60 gün gibi çok daha kısa bir TTL ayarlanacak. Bronze verisi "veri ambarının sigortasıdır" ancak sonsuza kadar aktif tutulması gerekmez. Bu otomatik silme işlemi, toplam depolama maliyetini ciddi oranda düşürür. (Gold katmanındaki analitik tablolara dokunulmaz, onlar kalıcıdır).

2.2. İşlem (Compute/Query) Maliyet Optimizasyonu

- **dbt Incremental Models :** Bu, işlem maliyetindeki en büyük kazanç yöntemidir. Silver ve Gold katmanlarındaki fct_ (Fact) tabloları, dbt'nin materialized: incremental (özelliği kullanılarak geliştirilecek. Günlük 100 milyon satır verinin tamamını (örneğin 30 milyar satırlık bir tabloyu) her gün yeniden işlemek (full-refresh) hem çok yavaş hem de aşırı pahalıdır. Artımlı model sayesinde, dbt sadece son çalışmadan bu yana eklenen *yeni* 100 milyon satırı işler. Bu, işlem maliyetini %90-99 oranında azaltır.
- **Fiyatlandırma Modeli :** Bu ağırlıkta bir ETL iş yükü için "On-Demand" (kullandıkça öde / taradığın TB başına ücret) fiyatlandırması yerine BigQuery Editions (Flat-Rate / Sabit Ücret) modeline geçiş planlanabilir. On-Demand modeli, dbt gibi ağır ve düzenli ETL işlemlerinde fatura sürprizlerine yol açabilir. Sabit (Flat-Rate) model ise size aylık öngörülebilir bir bütçeyle adanmış işlem kapasitesi sunar. Ağır ETL işleri için bu model neredeyse her zaman daha ucuzdur.
- **dbt Staging Modelleri (View/Ephemeral):** Silver katmanındaki ilk stg_ (staging) modelleri (örn: un-nesting veya CAST yapan modeller) fiziksel bir tablo (materialized: table) olarak oluşturulmayacak. Bunun yerine materialized: view (görünüm) veya ephemeral (geçici/sanal) olarak ayarlanacak. Her küçük ara adım için fiziksel tablo oluşturmak gereksiz depolama maliyeti yaratır. ephemeral modeller dbt tarafından otomatik olarak bir CTE (Common Table Expression) bloğuna dönüştürülür, böylece maliyet yaratmazlar.

- **Sorgu Optimizasyonu :** Tüm dbt modellerinde ve BI (Looker Studio) araçlarında SELECT * kullanımından kesinlikle kaçınılacak. Sadece ihtiyaç duyulan sütunlar seçilecek. BigQuery sütun bazlı (columnar) bir veritabanıdır ve maliyetlendirme, sorgunun *taradığı* sütunlara göre yapılır. 100 sütunlu bir tablodan SELECT * çekmek, 3 sütun seçmekten 33 kat daha pahalıdır. Bu kuralı dbt'de uygulamak, tüm DWH'in daha verimli çalışmasını sağlar.

3. Veri Yükleme Yöntemleri ve Süreçleri

3.1. Kaynaktan Taşıma (Source -> Bronze)

- Süreç: GA4'ten BigQuery'ye (Bronze) Ham Veri Aktarımı
- Yöntem: Google Analytics 4 "BigQuery Export"
- Açıklama:
    - Bu, kod gerektirmeyen ve en verimli yöntemdir. Herhangi bir script (örn: Dataflow) yazmamıza gerek kalmaz.
    - GA4 arayüzünden, verinin aktarılacağı GCP projesindeki BigQuery servisi linklenir.
    - Export tipi olarak Daily seçilir.
    - Bu kurulumun ardından Google, süreci tam otomatik olarak yönetir. Her gün, bir önceki günün tüm verilerini (bizim senaryomuzda 100 milyon satır) toplar ve BigQuery'deki ilgili dataset'e events_YYYYMMDD adında yeni bir tablo olarak aktarır.
    - Bu events_YYYYMMDD tabloları, üzerinde hiçbir değişiklik yapılmamış, ham JSON verisini içeren Bronze katmanımızı oluşturur.

3.2. Layer'lar Arası Hesaplama ve Taşıma

Süreç, veri BigQuery'ye *girdikten sonra* başlar. Veriyi temizlemek (Silver) ve iş metriklerine dönüştürmek (Gold) için dbt'yi kullanırız. Bu bir ELT yaklaşımıdır; veriyi BQ dışına *çıkarmadan* BQ'nin kendi işlem gücüyle dönüştürürüz.

- Süreç: Medallion Katmanları Arası Veri Dönüşümü
- Yöntem: dbt (Data Build Tool) + BigQuery İşlem Gücü
- Açıklama: Tüm iş mantığı (temizleme, un-nesting, Star Schema'ya dönüştürme) dbt projemizde SQL (.sql dosyaları) olarak tanımlanır.
    - **Bronze -> Silver (Temizleme & Yapılandırma):**
        - dbt, Bronze katmanındaki events_YYYYMMDD tablolarını kaynak olarak okur.
        - İlk olarak JSON'ları açmak (UNNEST), veri tiplerini düzeltmek (CAST) ve temel iş kurallarını (örn: session_id olmayanları ayıklama) uygulamak için stg_events, stg_sessions gibi Silver modellerini çalıştırır.
        - **Önemli Not:** 2. maddede konuştuğumuz maliyet optimizasyonu burada devreye girer. Bu modeller artımlı (incremental) olarak çalışır. dbt, her gün sadece *yeni gelen* events_YYYYMMDD tablosundaki 100 milyon satırı işler, geçmişteki milyarlarca satırı tekrar işlemez.

    - İşlenen temiz veri, BigQuery'deki silver dataset'ine stg_events tablosu olarak (veya view olarak) yazılır.
- **Silver -> Gold (Modelleme & Aggregasyon):**
    - Silver katmanındaki temizlenmiş veriler (örn: stg_events) dbt tarafından kaynak olarak okunur.
    - dbt, bu verileri kullanarak Star Schema mantığındaki son kullanıcı tablolarını (fct_ecommerce_transactions, dim_user, dim_device vb.) oluşturur.
    - dim_ (Dimension) tabloları, verinin yapısına göre (örn: dim_device) her gün yeniden hesaplanabilir (materialized) veya yavaş değişen (SCD Type 2) artımlı modeller olabilir.
    - fct_ (Fact) tabloları ise kesinlikle incremental olmalıdır. Her gün sadece yeni gelen purchase event'leri fct_ecommerce_transactions tablosuna eklenir.
    - Bu hesaplamaların sonucu, BigQuery'deki gold (veya analytics) dataset'ine analistlerin kullanacağı son tablolar olarak yazılır.

3.3. Süreç Zamanlama (Schedule)

- Süreç: Günlük DWH (ETL) Orkestrasyonu
- Yöntem: Cloud Composer (Managed Airflow)
- Açıklama: Tüm bu sürecin otomatik, sıralı ve hata kontrolü yapılarak çalışmasını Airflow sağlar. Bunun için bir DAG oluşturulur:
    - Görev 1 (Sensor): Günlük GA4 verisinin BigQuery'ye (Bronze) düştüğünü kontrol eden bir BigQueryTableSensor kullanılır. Bu sensör, events_YYYYMMDD (o güne ait) tablosu Bronze katmanında oluşana kadar bekler.
    - Görev 2 (dbt Run - Silver): Sensör başarılı olduğunda (yani veri geldiğinde), Airflow bir BashOperator (veya DbtRunOperator) kullanarak dbt'nin Silver modellerini tetikler (örn: dbt run --models tag:silver).
    - Görev 3 (dbt Run - Gold): Görev 2 başarıyla bittiğinde, Gold katmanını besleyen modeller tetiklenir (örn: dbt run --models tag:gold). Bu bağımlılık (Görev 2 >> Görev 3) Silver bitmeden Gold'un çalışmamasını garanti eder.
    - Görev 4 (dbt Test): Gold modelleri oluştuktan sonra, veri kalitesini doğrulamak için dbt test komutu çalıştırılır.
    - Görev 5 (Bildirim): DAG tamamlandığında veya herhangi bir adımda (örn: dbt test fail ederse) hata verdiğinde, SlackWebhookOperator veya EmailOperator ile ilgili ekibe (size) bildirim gönderilir.

4. Test ve Validasyon Süreçleri

4.1. dbt Testleri

- **Schema Testleri:** Bunlar, model sütunlarına uyguladığım hızlı ve standart testlerdir. .yml dosyaları içinde tanımlanırlar.
    - **not_null :** Gold katmanındaki fct_ tablolarının anahtar sütunları (örn: fct_ecommerce_transactions.transaction_id) veya dim_ tablolarının user_key, date_key gibi anahtarları asla NULL olmamalıdır.
    - **unique :** Olgusal (fact) tabloların grain'ini (tanecik yapısını) korumak için fct_ecommerce_transactions.transaction_id sütununun benzersiz olduğunu garanti ederim. Benzer şekilde dim_user.user_key de benzersiz olmalıdır.
    - **relationships :** Bu, Star Schema için önemli bir testtir. fct_ecommerce_transactions tablosundaki her user_key'in, dim_user tablosunda bir karşılığı (user_key) olduğunu doğrularım. Bu testi tüm Fact-Dimension ilişkileri (FK->PK) için kurarım. Bu sayede boşa düşen kayıtların oluşmasını engellerim.
    - **accepted_values :** Örneğin dim_device.device_category sütununun sadece 'mobile', 'desktop', 'tablet' değerlerini (veya NULL) alabileceğini doğrularım. 'Mobile' (büyük harfle) gibi beklenmedik bir değer gelirse test başarısız olur ve hatayı anında yakalarım.
- **Data Testleri :** Bunlar, benim özel olarak .sql formatında yazdığım ve *sıfır satır döndürmesi gereken* SQL sorgularıdır.
    - **Metrik Doğrulaması:** fct_ecommerce_transactions tablosundaki purchase_revenue_in_usd metriğinin *negatif* olamayacağını test ederim.
    - **Artımlı Yükleme (Incremental Load) Kontrolü:** Artımlı çalışan fct_ tablolarının, "dün" çalıştırıldığında, event_timestamp'i "dünden" (veya son 2 günden) eski olan bir veriyi işlememiş olduğunu doğrularım. Bu, artımlı yükleme mantığımın doğru çalıştığını garanti eder.
    - **İş Mantığı Tutarlılığı:** A tablosundaki toplam gelir ile B tablosundaki kırılımlı gelirlerin birbirini tuttuğunu kontrol eden bir sorgu yazabilirim (Örn: fct_item_sales toplamı, fct_ecommerce_transactions toplamı ile eşleşiyor mu?).

Bu testler nasıl çalışır? 3. madde'de bahsettiğim Airflow DAG'inin son adımı (Görev 4) olarak dbt test komutunu eklerim. Eğer dbt modelleri (Silver/Gold) başarıyla oluşsa bile, bu testlerden *herhangi biri* başarısız olursa (örn: transaction_id'de duplike veri bulunursa), Airflow DAG'i "failed" (başarısız) durumuna geçer ve bana (ve ekibe) anında alarm gönderir.

4.2. Kaynak Veri Validasyonu

- **Freshness Testi:** dbt'nin dbt source freshness özelliğini kullanırım. Bu komut, Bronze katmanındaki events_YYYYMMDD tablosunun "dün" için beklendiği gibi (örn: son 24 saat içinde) oluşup oluşmadığını kontrol eder. Eğer GA4 -> BQ export'unda bir sorun olursa, bu test başarısız olur ve DWH sürecinin (dbt run) hiç başlamadan durmasını sağlarım.

- **Satır Sayısı Anomali Tespiti:** Günlük 100 milyon satır veri beklerken, bir gün aniden 1 milyon satır veya 500 milyon satır gelmesi bir anomali işaretidir. Kaynak tablo (events_YYYYMMDD) üzerinde basit bir satır sayısı kontrolü (veya elementary-data gibi bir dbt eklentisi kullanarak) bu ani değişimleri yakalayıp alarm üretebilirim.

4.3. Manuel Validasyon

Tüm otomatik testler geçse bile, son verinin "mantıklı" olup olmadığını kontrol etmek gerekir.

- **Dashboard Karşılaştırması:** Gold katmanından beslenen Looker Studio dashboard'undaki "Dünkü Toplam Gelir" metriğini, doğrudan GA4 arayüzündeki (veya varsa eski bir sistemdeki) "Dünkü Toplam Gelir" metriği ile periyodik olarak karşılaştırırım. Bu, tüm pipeline'ın iş birimleri için doğru sonucu ürettiğinin nihai teyididir.
- **Smoke Test:** dbt pipeline'ı başarıyla bittikten sonra, Gold katmanındaki dim_ ve fct_ tablolarına SELECT ... LIMIT 10 gibi basit sorgular atarak verinin *göründüğünü* ve yapısının bozuk olmadığını (örn: tüm sütunların NULL gelmediğini) hızlıca kontrol edebilirim.

5. Data Science için DataMart Oluşturma

5.1. Katmanlı Yapı ve Mimari

- Yeni Katman: Data Science Mart (veya Feature Store) Katmanı, benim mimarimde bu, Gold katmanından *sonra* gelen bir katman olacak.
  Mimari Akışı: Bronze (Ham) -> Silver (Temiz) -> Gold (BI - Star Schema) -> DS-Mart
- Bu Katmanın Amacı: Star Schema, veriyi normalize eder (Fact ve Dimension'lara ayırır). DS ekibinin ihtiyacı ise bunun tam tersidir: de-normalize edilmiş, user_pseudo_id veya session_id bazında gruplanmış, tüm özelliklerin (features) tek bir satırda toplandığı geniş (wide) tablolar oluşturmaktır.
- **Yapı:**
  - **Kaynak:** Gold katmanındaki fct_ ve dim_ tabloları.
  - **İşlem:** JOIN işlemleri, PIVOT işlemleri (örn: event adlarını sütunlara çevirme) ve "Feature Engineering" (Örn: session_duration_sec / pageview_count = avg_time_per_page).
  - **Çıktı:** BigQuery'de ds_marts adında yeni bir dataset.
  - **Örnek Tablo:** user_session_features adında bir tablo. Bu tabloda her satır bir session_id olur ve o oturuma ait *tüm* bilgiler (kullanıcının cihazı, ülkesi, trafik kaynağı, oturumdaki pageview sayısı, click event'lerinin sayısı, purchase yapıp yapmadığı vb.) tek bir satırda sütun olarak yer alır.

Bu yapı, DS ekibinin tekrar tekrar JOIN yazmasını engeller ve BI sorgularının (Gold) DS sorgularından (DS-Mart) ayrışmasını sağlar.

5.2. Kullanılacak Teknolojiler ve Görevleri

Bu DS-Mart katmanını oluşturmak için mevcut teknoloji yığınımızı kullanmaya devam edeceğim:

- **BigQuery:** Hem Gold katmanındaki kaynak veriyi okuyacağımız yer, hem de DS-Mart tablolarını depolayacağımız yerdir. Tüm işlem gücü yine BQ üzerinde olacaktır.
- **dbt:** Bu iş için ana aracımızdır**.** DS ekibinin ihtiyacı olan o geniş user_session_features tablosu, özünde dbt için yeni bir modeldir. models/ds_marts/user_session_features.sql adında bir .sql dosyası oluştururum. Bu SQL, Gold katmanındaki fct_session_summary, dim_user, dim_device vb. tabloları JOIN'leyerek o geniş tabloyu oluşturur.
- **Cloud Composer (Airflow):** Orkestrasyonu sağlar. Ana DWH DAG'inde Gold katmanı bittikten *sonra* çalışacak şekilde dbt run --models tag:ds_mart komutunu tetikleyen yeni bir task eklerim. Böylece DS Mart'ları, Gold verisi tazelendikten hemen sonra otomatik olarak güncellenir.
- **Vertex AI (Gelecek Adımı):** DS ekibi bu DS-Mart tablosunu model eğitimi için doğrudan kullanabilir. Eğer bu özelliklerin (features) anlık (real-time) sunulması gerekirse, dbt'nin oluşturduğu bu tablodan beslenen bir Vertex AI Feature Store pipeline'ı da bu mimarinin bir sonraki adımı olarak eklenebilir.

5.3. Süreçlerin CI/CD Tool'ları

Tüm bu süreçleri (hem DWH hem de DS-Mart) otomatikleştirmek için standart bir DevOps CI/CD yapısı kurgularım.

- **Kod Deposu (Source Control):** GitHub. Tüm dbt SQL kodları ve Airflow DAG (Python) kodları burada yaşar. "Main" branch'i production ortamını temsil eder.
- **CI/CD Pipeline Aracı:** GCP Cloud Build

CI Süreci (Bir "Pull Request" açıldığında):

- **dbt için:** Cloud Build tetiklenir, dbt compile (SQL/Jinja syntax hatası var mı?) ve dbt test komutlarını *dev* ortamında çalıştırır.
- **Airflow için:** Cloud Build tetiklenir, Python flake8 (linting) kontrolü ve DAG import testleri yapar.

CD Süreci (Kod main-branch'ine merge edildiğinde):

- **dbt için:** Bu süreçte "deploy" edilecek bir uygulama sunucusu yoktur. Deploy etmek, kodun main-branch'inde olmasıdır. Airflow zaten her çalıştığında en güncel main-branch'ini kullanır (eğer Cloud Composer'ı Git-Sync ile kurduysak).
- **Airflow için:** Yeni veya güncellenmiş DAG (.py) dosyası, Cloud Build tarafından otomatik olarak Cloud Composer'ın GCS bucket'ındaki dags/ klasörüne kopyalanır. Airflow bu klasörü otomatik taradığı için DAG anında güncellenir.

5.4. Airflow için Test Süreçleri

Airflow DAG'lerinin kalitesi, pipeline'ın sağlığı için kritiktir. Testleri üçe ayırırım:

- **Statik Test (CI sırasında):**
    - **Linting:** flake8 gibi bir linter ile Python kod standartlarını kontrol ederim.
    - **Import Testi:** DAG dosyasının python my_dag.py komutuyla import edilip edilemediğini test ederim.
    - **Requirements:** requirements.txt dosyasının güncel ve eksiksiz olduğunu kontrol ederim.
- **Unit Test:**
    - **DAG Yapı Testi:** DAG'in temel özelliklerini (doğru schedule_interval ayarlanmış mı? retry sayısı doğru mu? owner bilgisi var mı?) test ederim.
    - **Döngü Testi :** DAG'in içinde task1 >> task2 >> task1 gibi bir döngü (cycle) olup olmadığını test ederim. Airflow'un kendi test kütüphaneleri bunu sağlar.
- **Staging Test:**
    - prod ortamının bir kopyası olan bir **Staging Cloud Composer** ortamı kurarım.
    - Yeni bir DAG'i main-branch'ine merge etmeden önce, Staging Airflow'da çalıştırırım.
    - Bu DAG, veriyi Staging BQ dataset'lerine (üretim verisinin küçük bir kopyası) yazar.
    - DAG'in baştan sona (Sensor -> dbt run silver -> dbt run gold -> dbt test -> dbt run ds_mart) başarıyla çalışıp çalışmadığını *gerçek* ortamda gözlemlerim.

5.5. SQL (dbt) Tarafında CI/CD Süreçleri

- **Geliştirme:** feature/add-new-dim adında yeni bir branch açarım. Yeni dbt modelimi (.sql) ve testlerimi (.yml) eklerim.
- **Pull Request (PR) Açma:** Kodumu main-branch'ine birleştirmek için bir PR açarım.
- **CI Pipeline'ı (Otomatik Tetikleme - örn: Cloud Build):**
    - **Adım 1: Kod Stili Kontrolü:** sqlfluff gibi bir SQL linter çalıştırarak kodun standartlara uymasını sağlarım.
    - **Adım 2: dbt Derleme:** dbt compile çalıştırırım. Bu, SQL veya Jinja syntax hatası olup olmadığını kontrol eder.
    - **Adım 3: dbt Çalıştırma (Dev Ortamda):** PR'a özel geçici bir BQ dataset'i (pr_123_dataset) oluştururum. dbt build --select my_changed_model+ komutunu çalıştırırım. Bu, *sadece benim değiştirdiğim modeli ve ondan etkilenen aşağı yönlü modelleri* bu geçici dataset'te oluşturur.
    - **Adım 4: dbt Test Etme:** dbt test --select my_changed_model+ komutunu çalıştırırım. Bu, yeni oluşturduğum modelin not_null, unique, relationship gibi testlerini bu geçici dataset üzerinde çalıştırır.
- **Onay:** Eğer tüm bu adımlar (lint, compile, build, test) başarılı olursa, CI pipeline'ı "yeşil" (başarılı) olur. Kodum teknik olarak "doğrududur" ve bir ekip arkadaşım tarafından incelenip main-branch'ine merge edilebilir.
- **CD (Deployment):** Kod main-branch'ine merge edildiği anda, source of truth güncellenmiş olur. Bir sonraki zamanlanmış Airflow çalışmasında (schedule_interval), Airflow bu yeni (main'deki) dbt kodunu kullanarak üretim DWH'ini günceller.

5.6. Sorgu Yazımında Dikkat Edilmesi Gerekenler ve Örnek Execution Plan

Sorgu Yazım Kuralları (Maliyet ve Performans):

- SELECT * kesinlikle kullanılmamalı: BigQuery sütun bazlı (columnar) çalışır. SELECT * yapmak, 100 sütunlu bir tablodan sadece 2 sütuna ihtiyacınız olsa bile, 100 sütunun tamamını tarar ve 50 kat fazla maliyet çıkarır. Her zaman SELECT user_key, device_category gibi net sütunlar seçilmelidir.
- PARTITION ve CLUSTER sütunlarını filtre edilmesi: Bu en önemli kuraldır. Sorgularda *mutlaka* WHERE event_date BETWEEN '2025-10-01' AND '2025-10-31' gibi bir filtre olmalıdır. Bu, BigQuery'nin tüm tabloyu değil, sadece ilgili 31 partition'ı taramasını sağlar. Benzer şekilde user_key (cluster) filtresi eklemek performansı katlar.
- GROUP BY optimizasyonu: GROUP BY yaparken user_key (cluster) veya date_key (partition) sütunlarını kullanmak, verinin fiziksel yerleşimine uyduğu için çok hızlı çalışır.
- JOIN optimizasyonu**:** Büyük tabloları (fct_) birbirine JOIN'lemekten kaçınılmalıdır. Star Schema modelimiz (Fact -> Dim) zaten bunu engeller. Her zaman büyük tabloyu (Fact) küçük tablolara (Dim) JOIN'lemeliyiz.
- APPROX fonksiyonlar**:** Benzersiz kullanıcı sayısını saymak için COUNT(DISTINCT user_pseudo_id) çok pahalı bir işlemdir. Eğer %100 doğruluk gerekmiyorsa, APPROX_COUNT_DISTINCT(user_pseudo_id) kullanmak binlerce kat daha hızlı ve ucuzdur.

Örnek SQL Execution Planı

Geliştirdiğim Star Schema modeline (fct_ecommerce_transactions ve dim_user) dayalı bir sorgu ve onun basitleştirilmiş Execution Planı:

Analist Sorusu: "Son 30 günde, Google'dan gelen ve mobil cihaz kullanan kullanıcıların günlük toplam geliri ne kadardır?"

Örnek Optimize SQL:

```
SELECT
  t.date_key,
  SUM(t.purchase_revenue_in_usd) AS total_revenue
FROM project.gold.fct_ecommerce_transactions AS t
JOIN project.gold.dim_session AS s ON t.session_key = s.session_key
JOIN project.gold.dim_traffic_source AS ts  ON s.traffic_key = ts.traffic_key
JOIN project.gold.dim_device AS d ON s.device_key = d.device_key
WHERE
  t.date_key >= 20251010 -- Partition
  AND ts.traffic_source_source = 'google'
  AND d.device_category = 'mobile'
GROUP BY
  t.date_key
ORDER BY
  t.date_key DESC;
```

BigQuery Execution Plan:

BigQuery bunu aşağıdan yukarıya okur:

Stage 0: Input (Dimension'lar)

- S00: INPUT (dim_traffic_source): dim_traffic_source tablosunu okur.
- S01: FILTER: traffic_source_source = 'google' filtresini uygular.
- S02: INPUT (dim_device): dim_device tablosunu okur.
- S03: FILTER: device_category = 'mobile' filtresini uygular. (Çok az satır döner).
- S04: INPUT (dim_session): dim_session tablosunu okur.

Stage 1: Dimension'ları Birleştirme (JOIN)

- S05: JOIN (Broadcast): dim_session (S04) ile filtrelenmiş dim_traffic_source (S01) sonucunu traffic_key üzerinden birleştirir
- S06: JOIN (Broadcast): Önceki join sonucu (S05) ile filtrelenmiş dim_device (S03) sonucunu device_key üzerinden birleştirir.
- **Sonuç:** Bu stage sonunda elimizde, "Google'dan gelen mobil oturumların" session_key listesi vardır.

Stage 2: Fact Tabloyu Okuma ve Filtreleme (En Önemli Adım)

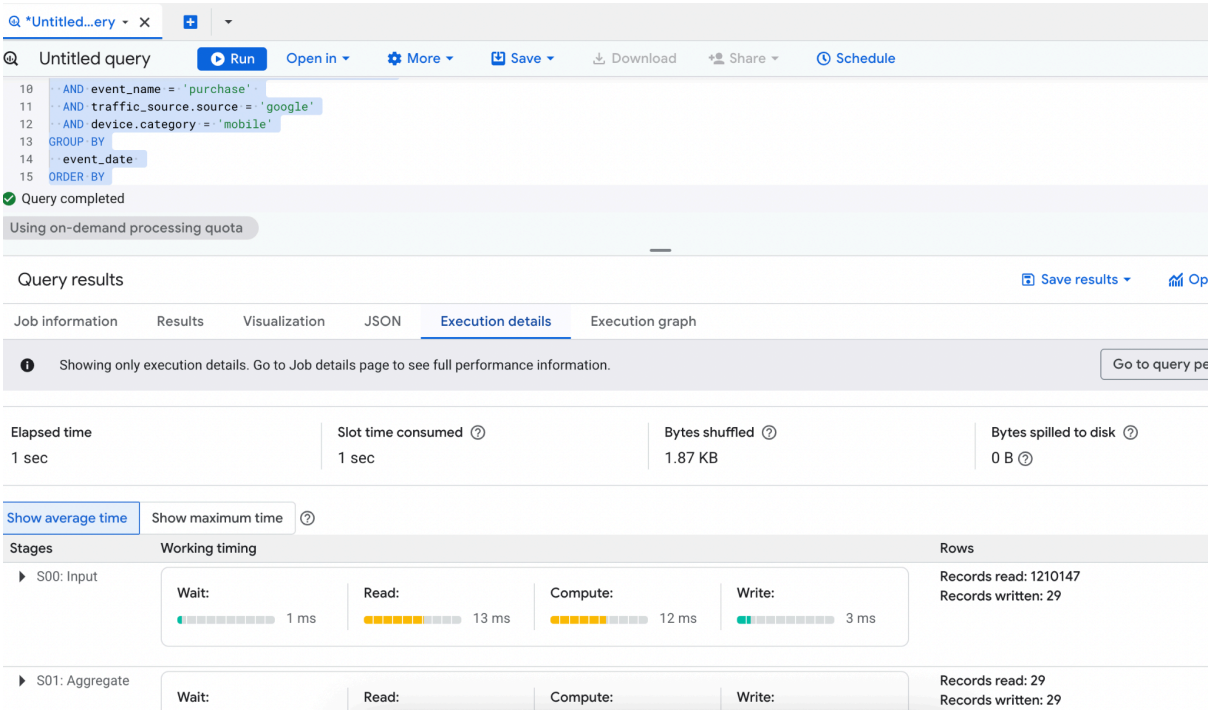
- S07: INPUT (fct_ecommerce_transactions): fct_ecommerce_transactions tablosunu okur.
- **PRUNING (Budama):** WHERE t.date_key >= 20251010 filtresi sayesinde BigQuery, bu tablonun **sadece son 30 günlük partition'larını (bölümlerini)** okur. Milyarlarca satırlık geçmiş veriye *dokunmaz*. **Maliyeti düşüren asıl adım budur.**

Stage 3: Final JOIN ve Aggregation

- S08: JOIN (Hash): Filtrelenmiş Fact tablosunu (S07) ile filtrelenmiş session_key listesini (S06) session_key üzerinden birleştirir (Hash Join, çünkü S07 hala büyüktür).
- S09: AGGREGATE: GROUP BY t.date_key ve SUM(t.purchase_revenue_in_usd) işlemlerini yapar.
- S10: SORT: ORDER BY t.date_key DESC ile sonucu sıralar.

Stage 4: Output

- S11: OUTPUT: Sonuçları kullanıcıya döndürür.


*Untitled...ery
Untitled query
Run
Open in
More
Save
Download
Share
Schedule
AND event_name = 'purchase'
AND traffic_source.source = 'google'
AND device.category = 'mobile'
GROUP BY
event_date
ORDER BY
Query completed
Using on-demand processing quota
Query results
Save results
Job information
Results
Visualization
JSON
Execution details
Execution graph
Showing only execution details. Go to Job details page to see full performance information.
Go to query pe
Elapsed time
1 sec
Slot time consumed
1 sec
Bytes shuffled
1.87 KB
Bytes spilled to disk
0 B
Show average time
Show maximum time
Stages
Working timing
Rows
S00: Input
Wait:
1 ms
Read:
13 ms
Compute:
12 ms
Write:
3 ms
Records read: 1210147
Records written: 29
S01: Aggregate
Wait:
Read:
Compute:
Write:
Records read: 29
Records written: 29