3.4. Entendendo os tipos de rotinas do Dataform e quando usar cada uma.

Considerações Gerais na discussão entre criar uma view ou uma tabela:

- **Dados Temporários vs. Persistentes:** Views são adequadas para transformações temporárias, enquanto tabelas nativas são preferíveis para armazenar dados transformados de forma persistente.
- **Complexidade da Lógica de Transformação:** Se a lógica de transformação é simples e não consome muitos recursos, uma view pode ser suficiente. Se a lógica é complexa, exigindo muitos recursos, uma tabela nativa pode ser mais apropriada.

Tipos de rotinas do Dataform quando usar e exemlos

1. “table” - Tabela nativa

Criar uma Tabela Nativa é quase sempre o melhor caso pelos ganhos de performance, reutilização e backup dos dados:

1. **Performance:**
    - **Caso de Uso:** Se a consulta da view for complexa e exigir recursos significativos durante a execução, pode ser mais eficiente criar uma tabela nativa para armazenar os resultados transformados.
    - **Vantagens:** A consulta de uma tabela nativa pode ser mais rápida do que a execução dinâmica de uma view complexa, especialmente para consultas frequentes.
2. **Persistência de Dados Transformados:**

- **Caso de Uso:** Quando você precisa persistir os resultados transformados para análises repetidas ou para uso em outras partes do pipeline.
- **Vantagens:** Armazenar os dados transformados em uma tabela nativa permite acesso rápido e repetido aos resultados sem a necessidade de recalcular as transformações a cada consulta.

3. **Particionamento e Clusterização:**
   - **Caso de Uso:** Se você precisa otimizar consultas por meio de particionamento e clusterização de tabelas, criar uma tabela nativa pode oferecer benefícios significativos.
   - **Vantagens:** Tabelas nativas podem ser particionadas e clusterizadas para melhorar o desempenho das consultas, especialmente quando há padrões de acesso específicos.

Exemplo de criação de uma tabela nativa pelo Dataform

```
config {
    type:"table",
    tags:["day", "aquisicao"],
    schema: dataform.projectConfig.vars.aquisicao_schema
}

WITH adjust_custos as (
  select
    day,
    campaign_id_network,
    partner,
    app_network,
    partner_id,
    app,
    partner_name,
    cost,
  from ${ref("tb_custos")} ac
  QUALIFY ROW_NUMBER() OVER (PARTITION BY campaign_id_network
)

SELECT
  ac.app as app_nome,
```

```
  ac.partner_name as canal_anuncio,
  DATE(ac.day) as dia,
  c.semana,
  c.mes,
  c.trimestre,
  c.ano,
  SUM(ac.cost) as custo
FROM adjust_custos ac
left join ${ref("calendario")} c on DATE(ac.day) = c.Data
group by
  ac.app,
  ac.partner_name,
  ac.day,
  c.semana,
  c.mes,
  c.trimestre,
  c.ano
```

2. “incremental” - Tabela incremental

Um processo de ETL incremental no BigQuery pode reduzir em até 99% o custo da rotina. Entenda quando usar as rotinas incrementais no dataform.

1. **Atualizações Frequentes:**
    - **Cenário:** Os dados fonte são atualizados ou modificados com frequência, mas apenas uma parte deles é alterada a cada vez.
    - **Vantagens:** O ETL incremental permite processar apenas os dados que foram adicionados ou modificados, reduzindo a carga de trabalho e melhorando a eficiência.
2. **Grandes Volumes de Dados:**
    - **Cenário:** Quando lidamos com grandes volumes de dados e processar o conjunto completo a cada vez é impraticável em termos de tempo e recursos.

- **Vantagens:** A abordagem incremental evita o processamento desnecessário de dados que não foram alterados, economizando recursos computacionais e melhorando a velocidade de execução do ETL.

3. **Custos de Processamento:**
   - **Cenário:** Se os custos associados ao processamento de dados no BigQuery são uma preocupação, a execução incremental pode ajudar a minimizar esses custos.
   - **Vantagens:** Ao processar apenas os dados modificados, você pode reduzir os custos relacionados ao processamento e ao armazenamento temporário durante o ETL.
4. **Atualização de Conjuntos de Dados Agregados:**
   - **Cenário:** Se você mantém conjuntos de dados agregados ou sumarizados e deseja atualizá-los regularmente.
   - **Vantagens:** Processar apenas os dados alterados permite que você mantenha os conjuntos de dados agregados atualizados sem a necessidade de reprocessar todo o conjunto de dados original.
5. **Aprimoramento do Desempenho:**
   - **Cenário:** Quando o desempenho é crítico e a latência precisa ser minimizada.
   - **Vantagens:** O ETL incremental reduz o tempo necessário para executar o processo, garantindo que apenas as alterações relevantes sejam processadas, o que é especialmente importante para aplicações em tempo real.

Exemplo de criação de uma tabela incremental pelo Dataform

```
config {
    type:"incremental",
    tags:["day","comunicacao"],
    bigquery: {
      partitionBy: "DATE(event_dt)",
      requirePartitionFilter: true,
    },
```

```
    protected: false,
    schema: dataform.projectConfig.vars.comunicacao_schema
}

WITH
  journey AS (
    SELECT
      id,
      name_nm,
      tags_nm
    FROM ${ref("tb_journey")}
    qualify row_number() over (partition by id order by creat
  )

,tbl as (
SELECT
  s.user_id,
  s.anonymous_id,
  s.event_nm,
  s.event_data_nm,
  s.category_nm,
  s.license_cod,
  s.event_dt,
  s.variation_id,
  s.id AS campaing_id,
  s.journey_id,
  LTRIM(LTRIM(J.name_nm, '[WEB]'), ' [Paliativo]') AS journey
  atualizacao_dt
FROM ${ref("tb_global_system_events")} S
LEFT JOIN journey J ON J.id = S.journey_id
WHERE J.tags_nm LIKE '%marketplace%' OR S.id IN ('104pb7e', '

)

select
  user_id,
  anonymous_id,
  event_nm,
```

```
  event_data_nm,
  category_nm,
  license_cod,
  event_dt,
  variation_id,
  campaing_id,
  journey_id,
  journey_nm,
  atualizacao_dt
 from tbl
${ when(incremental(),
  `WHERE atualizacao_dt >= timestamp_sub(current_timestamp(),
  AND atualizacao_dt > (SELECT MAX(atualizacao_dt) FROM ${sel
```

3. “incremental” - Snapshot

A lógica de snapshot de uma tabela no BigQuery é utilizada quando você precisa manter um histórico das mudanças nos dados ao longo do tempo. Em vez de apenas atualizar os dados existentes, a lógica de snapshot envolve criar novas linhas ou registros na tabela, cada uma representando um estado específico dos dados em um determinado ponto no tempo. Isso é útil para análises históricas, rastreamento de mudanças e auditoria. Aqui estão algumas situações em que a lógica de snapshot em uma tabela do BigQuery pode ser aplicada:

1. **Análises Temporais e Históricas:**
    - **Cenário:** Quando é necessário analisar dados em diferentes pontos temporais para entender como eles evoluíram ao longo do tempo.
    - **Uso:** Criar snapshots permite a realização de análises temporais, como a observação de tendências, variações sazonais e mudanças ao longo do tempo.
2. **Auditoria de Mudanças:**
    - **Cenário:** Quando é crucial manter um registro detalhado de todas as mudanças feitas nos dados.
    - **Uso:** Snapshots ajudam a auditar alterações, fornecendo uma trilha de auditoria que mostra quem fez quais alterações e quando elas ocorreram.

3. **Recuperação de Dados:**
    - **Cenário:** Em situações em que é necessário recuperar dados exatos a partir de um ponto específico no passado.
    - **Uso:** Ao manter snapshots, é possível restaurar a tabela para qualquer estado anterior, facilitando a recuperação de dados precisos em caso de erros ou eventos indesejados.
4. **Rastreamento de Dimensões em Modelos de Dados Dimensionais:**
    - **Cenário:** Em modelos de dados dimensionais, como estrela ou floco de neve, quando é necessário rastrear alterações nas dimensões ao longo do tempo.
    - **Uso:** Snapshots podem ser usados para manter históricos das dimensões, permitindo análises temporais mais precisas e rastreamento de mudanças nas características das dimensões.
5. **Consolidação de Dados de Fontes com Atualizações Incrementais:**
    - **Cenário:** Quando você integra dados de fontes externas que são atualizados incrementalmente e precisa manter um histórico consolidado.
    - **Uso:** A lógica de snapshot ajuda a consolidar dados de várias fontes, mantendo um histórico de todas as mudanças ocorridas em cada ponto no tempo.
6. **Manutenção de Histórico de Versões em Sistemas de Registro de Alterações:**
    - **Cenário:** Em sistemas que mantêm registros de alterações e é necessário manter um histórico de todas as versões.
    - **Uso:** Utilizando snapshots, é possível manter um histórico de versões detalhado, rastreando todas as mudanças feitas em um determinado registro.

Exemplo de criação de uma rotina de Snapshot pelo Dataform

```
config {
    type:"incremental",
    tags:["day","frota"],
    bigquery: {
```

```
      partitionBy: "DATE(atualizacaoData)",
      requirePartitionFilter: true,
    },
    protected: true,
    schema: dataform.projectConfig.vars.frota_schema
}

select
        id,
        criacaoData,
        idade_dias,
        km,
        placa,
        renavam,
        chassi,
        modelo,
        anoFabricacao,
        anoModelo,
        cor,
        veiculoModeloId,
        iotId,
        situacao_id,
        baseid,
        lugar_moto,
        pais_filial,
        descricao_situacao,
        data_situacao,
        dias_na_situacao,
        observacao,
        latitude,
        longitude,
        bateriaPrincipal,
        bateriaBackup,
        localizadorData,
        tempo_ultimo_ping_minutos,
        Idlocacao,
        situacao_locacao,
        tipo_situacao,
```

```
        tipo_situacao_detalhe,
        frota_operacional,
        usuarioId,
        regiao_nome,
        lugar_nome,
        base_latitude,
        base_longitude,
        lugar_Id,
        lugar_Idbase_gps,
        na_base,
        minhaMottu,
        usuario_teste,
        pacote_locacao,
        inicio_data_locacao,
        tempo_em_locacao_dias,
        dias_inadimplente,
        current_datetime("America/Sao_Paulo") as atualizacaoD
from ${ref("frota_atual")} fa
```

4. “view” - Tabela virtual

Criar uma View avalie se de fato é o melhor caso de uso:

1. **Transformações Leves ou Virtuais:**
   - **Caso de Uso:** Quando a transformação necessária é leve e pode ser expressa por meio de uma consulta SQL simples.
   - **Vantagens:** As views são consultas salvas que não ocupam espaço adicional de armazenamento. Elas oferecem uma maneira de criar "visualizações virtuais" dos dados sem a necessidade de armazenar duplicatas físicas.
2. **Reutilização de Lógica:**
   - **Caso de Uso:** Se uma determinada lógica de transformação é necessária em vários lugares, criar uma view permite a reutilização dessa lógica.

- **Vantagens:** Mudanças na lógica podem ser feitas em um único local (a definição da view), propagando automaticamente essas alterações para todos os lugares onde a view é referenciada.

3. **Economia de Espaço:**
   - **Caso de Uso:** Quando você deseja economizar espaço de armazenamento e não precisa de uma cópia física dos dados transformados.
   - **Vantagens:** As views não consomem espaço adicional, pois são consultas salvas que são executadas dinamicamente quando referenciadas.

Exemplo de criação de uma view pelo Dataform

```
config {
  type:"view",
  tags:["day","supply_chain"],
  schema:  dataform.projectConfig.vars.supply_chain_schema,
    description: `
        Descrição: Validação de Ajustes feitos pelas filiais (

        Unidade de negócio: Mottu Aluguel
        Área: Spare Parts
        Criado por: Victor Rodrigues
        Data da ultima atualização do código da tabela: 15/12
        Frequência de atualização: Daily
        Dados agrupados por: não agrupado
        Fuso horário das datas da tabela: Brazil UTC-03 `,
}


select
    dataCriacao,
    diaCriacao,
    Mes,
    Semana,
    LocalId,
    Local,
    OrderId,
    UserName,
```

```
    SkuId,
    MottuDescription,
    OriginalCode,
    Category,
    Situation,
    Units,
    StandardPrice,
    valor,
    InitialWarehouseStock,
    FinalWarehouseStock,
    tipoAjuste,
    itemLocalId,
    prctAjuste,
    Ajustavel,
    atualizacao_dt
 FROM ${ref("ajustes_estoque")}
WHERE diaCriacao > DATE(timestamp_sub(current_datetime("Ameri
```