Item 6 - Sobre Modelagem de Dados

Processo de Modelagem de Dados

Comecei entendendo o modelo relacional dos dados, compreendendo as tabelas, suas relações e como os dados são armazenados.

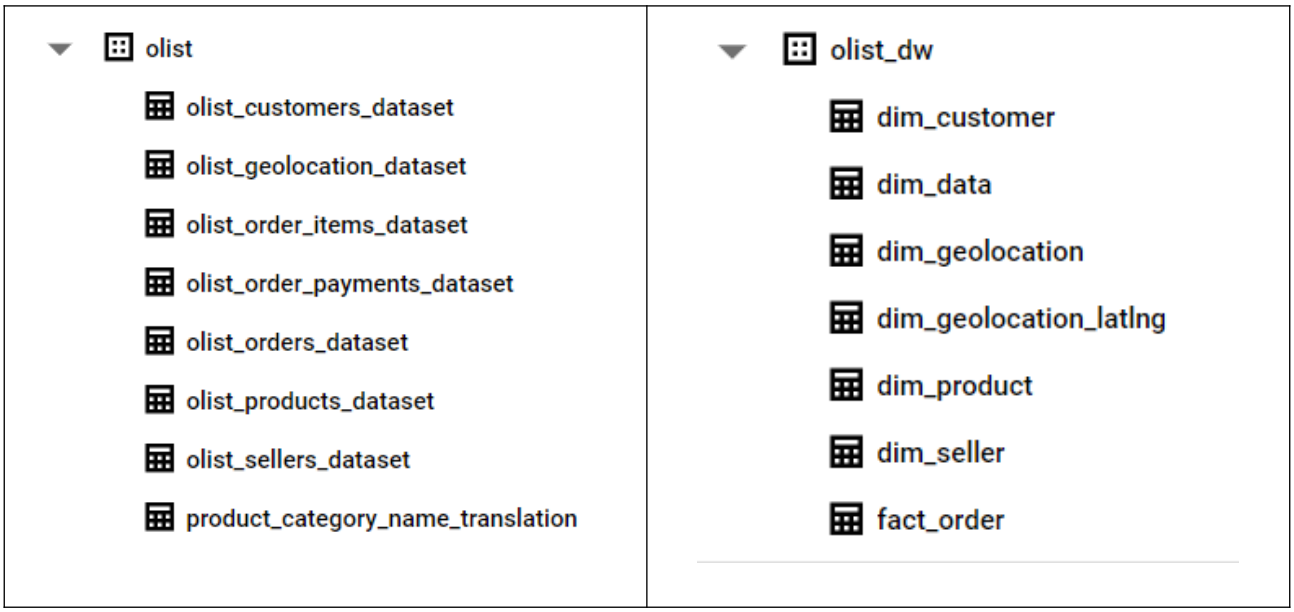
Em seguida, construí o modelo dimensional, criando tabelas de dimensão e fatos otimizadas para análise. As tabelas de dimensão contêm atributos descritivos, enquanto as tabelas de fatos armazenam as métricas a serem analisadas.

A próxima etapa foi construir consultas SQL para extrair dados das tabelas de origem e inseri-los nas tabelas de dimensão e fatos, garantindo a integridade dos dados e relacionamentos.

Esse processo poderia ser automatizado usando softwares de ETL, garantindo que o data warehouse esteja sempre atualizado.

As ferramentas de visualização tem um papel importante nesse processo de modelagem de dados pois podem a execução das consultas que respondem às perguntas de negócio, permitindo que os usuários finais interajam com os dados de forma fácil e intuitiva.

Para demonstrar a solução, importei os dados da Olist para o BigQuery e criei um conjunto de dados dimensional (olist_dw). As consultas SQL e o modelo dimensional podem servir de referência para outros com bases de dados similares.



Script para criar as tabelas de dimensão

```
CREATE TABLE `asteroide-attack-free-68204480.olist_dw.dim_customer` AS
SELECT
    GENERATE_UUID() as customer_sk,
    customer_id,
    customer_unique_id,
```

```
    customer_zip_code_prefix,
    customer_city,
    customer_state
FROM (
    SELECT DISTINCT
        customer_unique_id,
        ANY_VALUE(customer_id) as customer_id,
        ANY_VALUE(customer_zip_code_prefix) as customer_zip_code_prefix,
        ANY_VALUE(customer_city) as customer_city,
        ANY_VALUE(customer_state) as customer_state
    FROM `asteroide-attack-free-68204480.olist.olist_customers_dataset`
    GROUP BY customer_unique_id
);

-- dim_geolocation (sem latitude e longitude)
CREATE TABLE `asteroide-attack-free-68204480.olist_dw.dim_geolocation` AS
SELECT
    GENERATE_UUID() as geolocation_sk,
    geolocation_zip_code_prefix,
    geolocation_city,
    geolocation_state
FROM (
    SELECT DISTINCT
      geolocation_zip_code_prefix,
      ANY_VALUE(geolocation_city) as geolocation_city,
      ANY_VALUE(geolocation_state) as geolocation_state
    FROM `asteroide-attack-free-68204480.olist.olist_geolocation_dataset`
    GROUP BY geolocation_zip_code_prefix
);

-- dim_geolocation_latlng (apenas latitude e longitude)
CREATE TABLE `asteroide-attack-free-68204480.olist_dw.dim_geolocation_latlng` AS
SELECT
    GENERATE_UUID() as geolocation_latlng_sk,
    geolocation_zip_code_prefix,
    geolocation_lat,
    geolocation_lng
FROM (
    SELECT DISTINCT
      geolocation_zip_code_prefix,
      ANY_VALUE(geolocation_lat) as geolocation_lat,
      ANY_VALUE(geolocation_lng) as geolocation_lng
    FROM `asteroide-attack-free-68204480.olist.olist_geolocation_dataset`
    GROUP BY geolocation_zip_code_prefix
);

CREATE TABLE `asteroide-attack-free-68204480.olist_dw.dim_product` AS
SELECT
    GENERATE_UUID() as product_sk,
    product_id,
    product_category_name,
    product_name_lenght,
    product_description_lenght,
    product_photos_qty,
    product_weight_g,
    product_length_cm,
    product_height_cm,
    product_width_cm
FROM (
    SELECT DISTINCT
        product_id,
        ANY_VALUE(product_category_name) as product_category_name,
        ANY_VALUE(product_name_lenght) as product_name_lenght,
```

```
        ANY_VALUE(product_description_lenght) as product_description_lenght,
        ANY_VALUE(product_photos_qty) as product_photos_qty,
        ANY_VALUE(product_weight_g) as product_weight_g,
        ANY_VALUE(product_length_cm) as product_length_cm,
        ANY_VALUE(product_height_cm) as product_height_cm,
        ANY_VALUE(product_width_cm) as product_width_cm
    FROM `asteroide-attack-free-68204480.olist.olist_products_dataset`
    GROUP BY product_id
);

CREATE TABLE `asteroide-attack-free-68204480.olist_dw.dim_seller` AS
SELECT
    GENERATE_UUID() as seller_sk,
    seller_id,
    seller_zip_code_prefix,
    seller_city,
    seller_state
FROM (
    SELECT DISTINCT
        seller_id,
        ANY_VALUE(seller_zip_code_prefix) as seller_zip_code_prefix,
        ANY_VALUE(seller_city) as seller_city,
        ANY_VALUE(seller_state) as seller_state
    FROM `asteroide-attack-free-68204480.olist.olist_sellers_dataset`
    GROUP BY seller_id
);

CREATE OR REPLACE TABLE `asteroide-attack-free-68204480.olist_dw.dim_data` AS
SELECT
  GENERATE_UUID() as date_sk,
  date as data_completa,
  EXTRACT(YEAR FROM date) as ano,
  EXTRACT(MONTH FROM date) as mes,
  EXTRACT(DAY FROM date) as dia,
  EXTRACT(DAYOFYEAR FROM date) as dia_do_ano,
  EXTRACT(WEEK FROM date) as semana_do_ano,
  EXTRACT(DAYOFWEEK FROM date) as dia_da_semana,
  FORMAT_TIMESTAMP('%A', TIMESTAMP(date)) as nome_dia_da_semana,
  CASE
    WHEN EXTRACT(MONTH FROM date) BETWEEN 3 AND 5 THEN 'Primavera'
    WHEN EXTRACT(MONTH FROM date) BETWEEN 6 AND 8 THEN 'Verão'
    WHEN EXTRACT(MONTH FROM date) BETWEEN 9 AND 11 THEN 'Outono'
    ELSE 'Inverno'
  END as estacao
FROM
  UNNEST(GENERATE_DATE_ARRAY('2000-01-01', '2030-12-31')) as date;



CREATE OR REPLACE TABLE `asteroide-attack-free-68204480.olist_dw.fact_order` AS
SELECT
    oi.order_id,
    c.customer_sk,
    p.product_sk,
    s.seller_sk,
    oi.shipping_limit_date,
    oi.price,
    oi.freight_value,
    gc.geolocation_sk as customer_geolocation_sk,  -- Chave da dimensão de geolocalização do cliente
    gs.geolocation_sk as seller_geolocation_sk,   -- Chave da dimensão de geolocalização do vendedor
    d1.date_sk as order_purchase_date_sk,
    d2.date_sk as order_approved_at_sk,
    d3.date_sk as order_delivered_carrier_date_sk,
```

```
    d4.date_sk as order_delivered_customer_date_sk,
    d5.date_sk as order_estimated_delivery_date_sk
FROM `asteroide-attack-free-68204480.olist.olist_order_items_dataset` oi
LEFT JOIN `asteroide-attack-free-68204480.olist.olist_orders_dataset` o
    ON oi.order_id = o.order_id
LEFT JOIN `asteroide-attack-free-68204480.olist_dw.dim_customer` c
    ON o.customer_id = c.customer_id
LEFT JOIN `asteroide-attack-free-68204480.olist_dw.dim_product` p
    ON oi.product_id = p.product_id
LEFT JOIN `asteroide-attack-free-68204480.olist_dw.dim_seller` s
    ON oi.seller_id = s.seller_id
-- Junções com as dimensões de geolocalização
LEFT JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gc
    ON c.customer_zip_code_prefix = gc.geolocation_zip_code_prefix
LEFT JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gs
    ON s.seller_zip_code_prefix = gs.geolocation_zip_code_prefix
LEFT JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d1
    ON DATE(o.order_purchase_timestamp) = d1.data_completa
LEFT JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d2
    ON DATE(o.order_approved_at) = d2.data_completa
LEFT JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d3
    ON DATE(o.order_delivered_carrier_date) = d3.data_completa
LEFT JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d4
    ON DATE(o.order_delivered_customer_date) = d4.data_completa
LEFT JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d5
    ON DATE(o.order_estimated_delivery_date) = d5.data_completa;
```

Adendo sobre a Dimensão de Geolocalização, Otimização de Performance e Redução de Registros

Durante o processo de modelagem do data warehouse, identifiquei um desafio relacionado à dimensão de geolocalização (dim_geolocation). A presença de valores repetidos na coluna geolocation_zip_code_prefix, juntamente com a variação nos campos geolocation_city e geolocation_state para um mesmo CEP, impedia a criação de relacionamentos diretos e não ambíguos com a tabela de fatos (fact_order) em ferramentas de BI como o Power BI, que não suportam chaves primárias compostas.
Para solucionar essa limitação, garantir a integridade dos dados e otimizar o desempenho das consultas, adotei a seguinte estratégia:

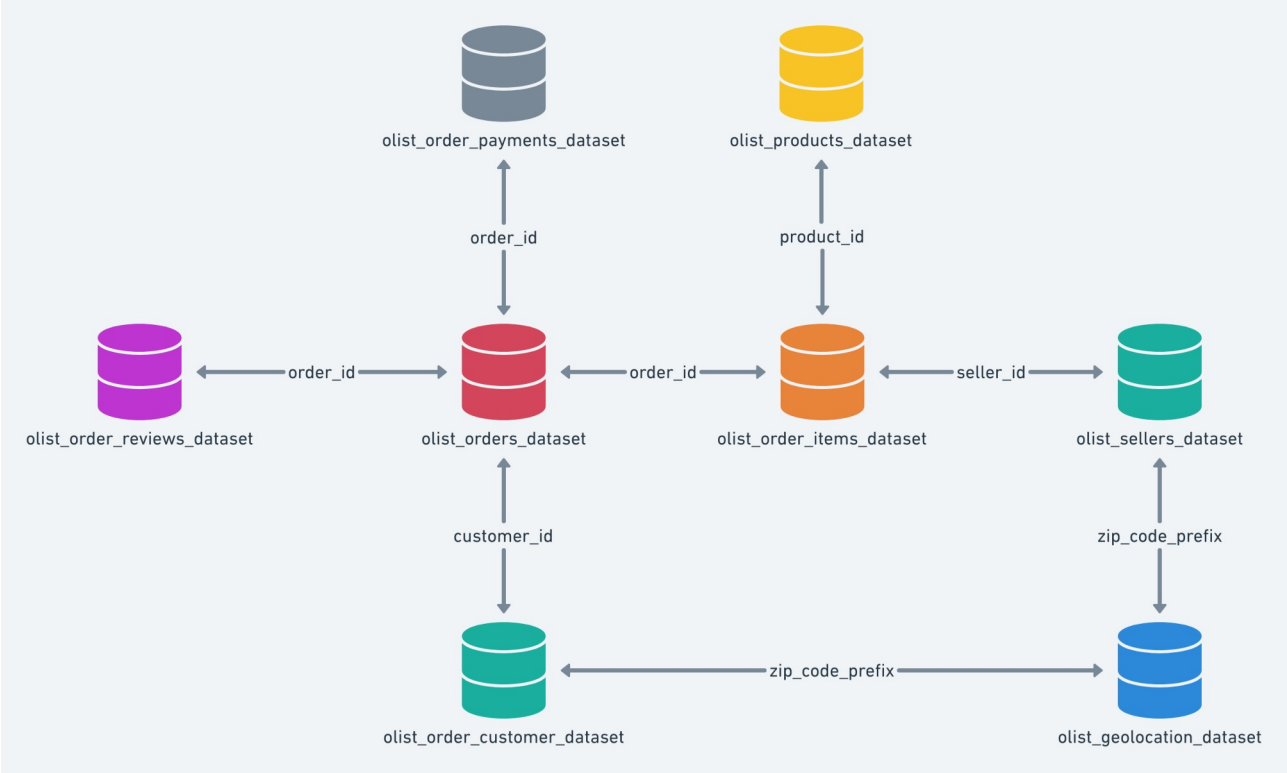
1. **Divisão da dimensão** dim_geolocation**:** A dimensão original foi dividida em duas tabelas:

- dim_geolocation: Contém informações sobre cidade e estado, utilizando geolocation_zip_code_prefix como chave primária. Essa dimensão é referenciada diretamente pela tabela de fatos através dos campos customer_zip_code_prefix e seller_zip_code_prefix.
- dim_geolocation_latlng: Contém informações sobre latitude e longitude, também utilizando geolocation_zip_code_prefix como chave primária. Essa dimensão está relacionada à

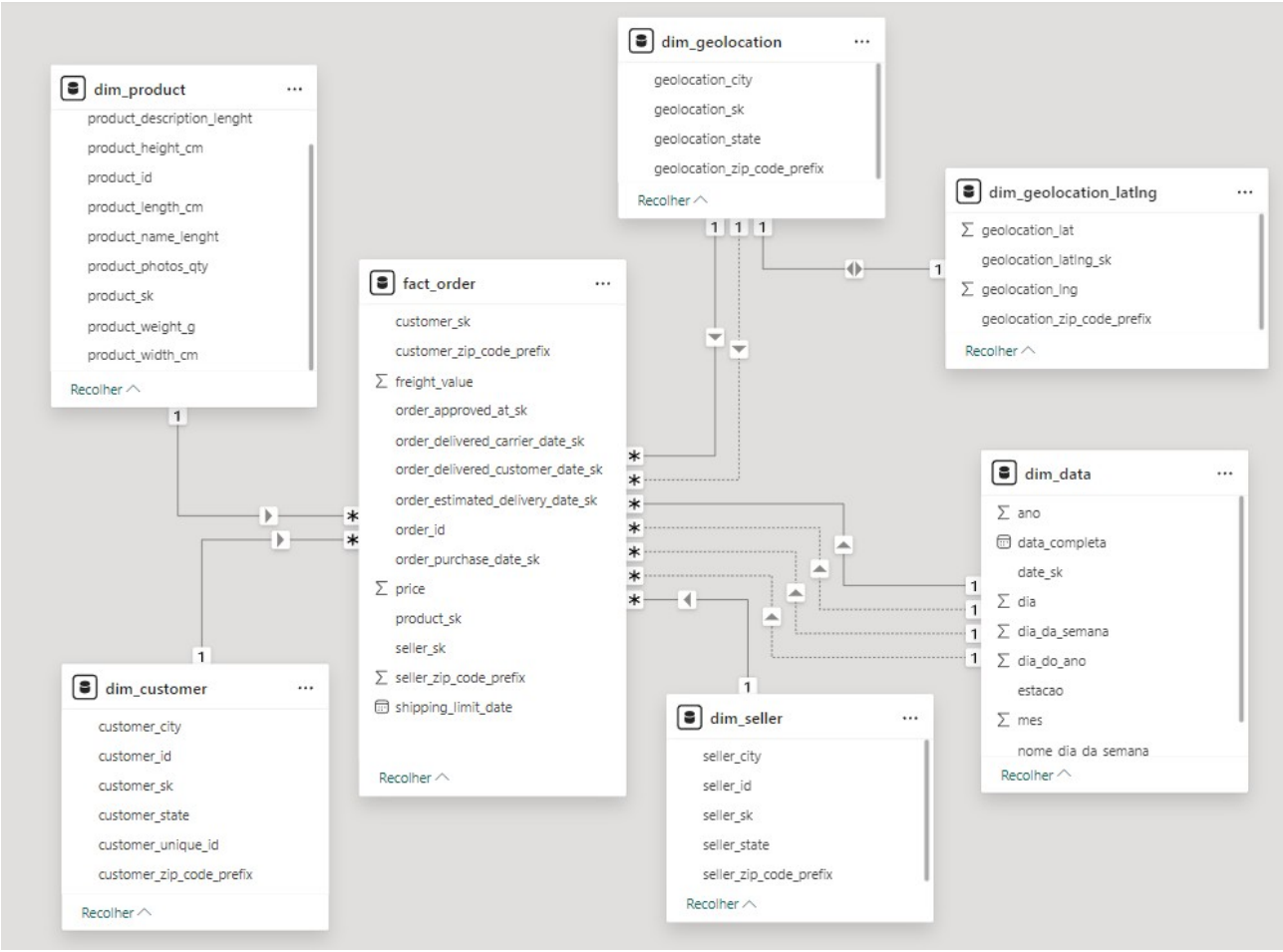
dim_geolocation através do campo geolocation_zip_code_prefix, permitindo o acesso às coordenadas geográficas quando necessário.

2. **Redução do número de registros na dimensão** dim_geolocation**:** Durante a criação da dim_geolocation, utilizei a cláusula GROUP BY geolocation_zip_code_prefix em conjunto com a função ANY_VALUE para selecionar apenas um registro representativo para cada CEP, mesmo que a tabela original contivesse múltiplos registros com o mesmo CEP e variações nos campos de cidade e estado. Essa abordagem permitiu reduzir o número de registros na dimensão de geolocalização de mais de 1 milhão para pouco mais de 19 mil, otimizando o desempenho das consultas e simplificando o modelo de dados.
   Essa estratégia resolve o problema da chave composta no Power BI, garante a integridade dos dados e oferece flexibilidade para realizar análises que envolvam a geolocalização de clientes e vendedores diretamente no data warehouse, sem depender de junções adicionais em ferramentas externas. Ao realizar as junções com as dimensões de geolocalização diretamente na query SQL que cria a tabela de fatos, evito a necessidade de realizar essas junções posteriormente em uma ferramenta de BI, otimizando ainda mais o desempenho e simplificando o modelo de dados.
   É importante ressaltar que a utilização da função ANY_VALUE implica na seleção de um valor arbitrário para os campos geolocation_city e geolocation_state caso haja variação desses valores para um mesmo CEP. Em meu caso, considerei essa seleção arbitrária aceitável, pois o foco principal da análise está na relação entre CEP e coordenadas geográficas. No entanto, caso a variação nos valores de cidade e estado para um mesmo CEP fosse relevante para essa análise, outras abordagens podem ser necessárias, como a criação de uma chave primária composta ou a utilização de uma coluna de granularidade mais fina na dimensão de geolocalização.

Modelagem relacional



Modelagem dimensional



Com essa modelagem, agora posso responder a várias perguntas de negócios, como análise de vendas ao longo do tempo, análise de desempenho do vendedor, análise de produtos mais vendidos, análise de comportamento do cliente, análise de tempo de entrega e análise de valor do frete.

No entanto, gostaria de mencionar que enfrentei algumas limitações durante esse processo. Devido a limitações no ambiente da Dadosfera e à falta de conhecimento, não foi possível importar todos os dados para a plataforma da Dadosfera. Mesmo que eu tivesse feito isso usando os arquivos CSV, ainda não seria possível fazer a modelagem de dados dentro da plataforma da Dadosfera.

Mas não se preocupem, eu encontrei uma solução. Para fazer demonstrações efetivas da solução, usei minha infraestrutura pessoal para criar, testar e validar o modelo de dados relacional. Aqui estão algumas consultas SQL que podemos usar para gerar insights a partir dos dados. Só essas queries já responderiam a várias perguntas do negócio, mas usar consultas sql não é algo escalável.

Perguntas que o DW pode responder com a inclusão das dimensões de geolocalização

Com a estrutura atualizada da tabela de fatos e as dimensões de geolocalização (dim_geolocation e dim_geolocation_latlng), o data warehouse pode responder às seguintes perguntas de negócio, tanto relacionadas a aspectos gerais de vendas quanto a análises geográficas específicas:

Perguntas Gerais:

- **Vendas Totais por Ano:**
    - Qual o total de vendas e número de pedidos por ano?
- **Vendas Totais por Vendedor:**
    - Qual o total de vendas e número de pedidos por vendedor?
- **Produtos Mais Vendidos:**
    - Quais são os produtos mais vendidos em termos de quantidade e valor total?
- **Análise de Tempo de Entrega:**
    - Qual o tempo médio de entrega dos pedidos em dias?
- **Análise de Valor do Frete:**
    - Qual o valor médio, máximo e mínimo do frete?

Análises Geográficas:

- **Vendas Totais por Região:**
    - Qual o total de vendas e número de pedidos por cidade, estado ou região?

- **Desempenho do Vendedor por Região:**
    - Qual o desempenho dos vendedores em diferentes regiões?
- **Produtos Mais Vendidos por Região:**
    - Quais são os produtos mais vendidos em cada região?
- **Tempo de Entrega por Região:**
    - Qual o tempo médio de entrega dos pedidos em diferentes regiões?
- **Valor do Frete por Região:**
    - Qual o valor médio do frete em diferentes regiões?
- **Comportamento do Cliente por Região:**
    - Existem preferências regionais em relação a determinados produtos ou categorias?
- **Distância Média entre Cliente e Vendedor:**
    - Qual a distância média entre a localização do cliente e do vendedor em cada pedido? (Utilizando dim_geolocation_latlng)
- **Regiões com Maior Concentração de Clientes e Vendedores:**
    - Quais são as regiões com maior número de clientes e vendedores?

Queries SQL para Perguntas Gerais:

1. **Vendas totais por ano**:

```
SELECT
    d.ano,
    COUNT(f.order_id) as total_orders,
    SUM(f.price) as total_sales
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d ON f.order_purchase_date_sk = d.date_sk
GROUP BY d.ano
ORDER BY d.ano;
```

2. **Vendas totais por vendedor**:

```
SELECT
    s.seller_id,
    COUNT(f.order_id) as total_orders,
    SUM(f.price) as total_sales
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_seller` s ON f.seller_sk = s.seller_sk
GROUP BY s.seller_id
ORDER BY total_sales DESC;
```

3. **Produtos mais vendidos**:

```
SELECT
    p.product_id,
    COUNT(f.order_id) as total_orders,
    SUM(f.price) as total_sales
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_product` p ON f.product_sk = p.product_sk
GROUP BY p.product_id
ORDER BY total_orders DESC;
```

4. **Análise de tempo de entrega**:

```
SELECT
    AVG(DATE_DIFF(d4.data_completa, d1.data_completa, DAY)) as average_delivery_days
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d1 ON f.order_purchase_date_sk = d1.date_sk
JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d4 ON f.order_delivered_customer_date_sk =
d4.date_sk;
```

5. **Análise de valor do frete**:

```
SELECT
    AVG(f.freight_value) as average_freight_value,
    MAX(f.freight_value) as max_freight_value,
    MIN(f.freight_value) as min_freight_value
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f;
```

Queries SQL para Análises Geográficas

1. Vendas Totais por Região:

```
-- Por cidade do cliente, incluindo a UF
SELECT
    gc.geolocation_city,
    gc.geolocation_state AS uf,
    COUNT(f.order_id) AS total_orders,
    SUM(f.price) AS total_sales
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gc
    ON f.customer_geolocation_sk = gc.geolocation_sk
GROUP BY gc.geolocation_city, gc.geolocation_state
ORDER BY total_sales DESC;

-- Por estado do cliente
SELECT
    gc.geolocation_state,
    COUNT(f.order_id) AS total_orders,
    SUM(f.price) AS total_sales
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gc
    ON f.customer_geolocation_sk = gc.geolocation_sk
GROUP BY gc.geolocation_state
ORDER BY total_sales DESC;
```

2. Desempenho do Vendedor por Região:

```
-- Por cidade do vendedor, incluindo a UF
SELECT
    gs.geolocation_city,
    gs.geolocation_state AS uf,
    s.seller_id,
    COUNT(f.order_id) AS total_orders,
    SUM(f.price) AS total_sales
```

```
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_seller` s
    ON f.seller_sk = s.seller_sk
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gs
    ON f.seller_geolocation_sk = gs.geolocation_sk
GROUP BY gs.geolocation_city, gs.geolocation_state, s.seller_id
ORDER BY gs.geolocation_city, total_sales DESC;

-- Por estado do vendedor
SELECT
    gs.geolocation_state,
    s.seller_id,
    COUNT(f.order_id) AS total_orders,
    SUM(f.price) AS total_sales
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_seller` s
    ON f.seller_sk = s.seller_sk
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gs
    ON f.seller_geolocation_sk = gs.geolocation_sk
GROUP BY gs.geolocation_state, s.seller_id
ORDER BY gs.geolocation_state, total_sales DESC;
```

3. Produtos Mais Vendidos por Região:

```
-- Por cidade do cliente, incluindo a UF
SELECT
    gc.geolocation_city,
    gc.geolocation_state AS uf,
    p.product_id,
    COUNT(f.order_id) AS total_orders,
    SUM(f.price) AS total_sales
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_product` p
    ON f.product_sk = p.product_sk
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gc
    ON f.customer_geolocation_sk = gc.geolocation_sk
GROUP BY gc.geolocation_city, gc.geolocation_state, p.product_id
ORDER BY gc.geolocation_city, total_orders DESC;

-- Por estado do cliente
SELECT
    gc.geolocation_state,
    p.product_id,
    COUNT(f.order_id) AS total_orders,
    SUM(f.price) AS total_sales
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_product` p
    ON f.product_sk = p.product_sk
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gc
    ON f.customer_geolocation_sk = gc.geolocation_sk
GROUP BY gc.geolocation_state, p.product_id
ORDER BY gc.geolocation_state, total_orders DESC;
```

4. Tempo de Entrega por Região:

```
-- Por cidade do cliente, incluindo a UF
SELECT
    gc.geolocation_city,
    gc.geolocation_state AS uf,  -- Adicionando a UF
    AVG(DATE_DIFF(d4.data_completa, d1.data_completa, DAY)) as average_delivery_days
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d1
    ON f.order_purchase_date_sk = d1.date_sk
JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d4
    ON f.order_delivered_customer_date_sk = d4.date_sk
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gc
    ON f.customer_geolocation_sk = gc.geolocation_sk
GROUP BY gc.geolocation_city, gc.geolocation_state  -- Agrupando também por estado
ORDER BY average_delivery_days DESC;

-- Por estado do cliente
SELECT
    gc.geolocation_state,
    AVG(DATE_DIFF(d4.data_completa, d1.data_completa, DAY)) as average_delivery_days
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d1
    ON f.order_purchase_date_sk = d1.date_sk
JOIN `asteroide-attack-free-68204480.olist_dw.dim_data` d4
    ON f.order_delivered_customer_date_sk = d4.date_sk
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gc
    ON f.customer_geolocation_sk = gc.geolocation_sk
GROUP BY gc.geolocation_state
ORDER BY average_delivery_days DESC;
```

5. Valor do Frete por Região:

```
-- Por cidade do cliente, incluindo a UF
SELECT
    gc.geolocation_city,
    gc.geolocation_state,
    AVG(f.freight_value) as average_freight_value
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gc
    ON f.customer_geolocation_sk = gc.geolocation_sk
GROUP BY gc.geolocation_city, gc.geolocation_state
ORDER BY average_freight_value DESC;

-- Por estado do cliente
SELECT
    gc.geolocation_state,
    AVG(f.freight_value) as average_freight_value
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gc
    ON f.customer_geolocation_sk = gc.geolocation_sk
GROUP BY gc.geolocation_state
ORDER BY average_freight_value DESC;
```

6. Comportamento do Cliente por Região:

```
-- Produtos mais comprados por cidade do cliente
SELECT
    gc.geolocation_city,
    p.product_category_name,
    COUNT(f.order_id) AS total_orders
FROM `asteroide-attack-free-68204480.olist_dw.fact_order` f
JOIN `asteroide-attack-free-68204480.olist_dw.dim_product` p
    ON f.product_sk = p.product_sk
JOIN `asteroide-attack-free-68204480.olist_dw.dim_geolocation` gc
    ON f.customer_geolocation_sk = gc.geolocation_sk
GROUP BY gc.geolocation_city, p.product_category_name
ORDER BY gc.geolocation_city, total_orders DESC;
```

Adendo: Pré-homologação do Dashboard e Potencial para Novos Insights

As queries SQL apresentadas representam uma pré-homologação das análises que poderão ser construídas na camada de visualização do nosso data warehouse. Com a modelagem dimensional implementada e a inclusão das dimensões de geolocalização, temos a capacidade de gerar diversos insights relevantes para o negócio, como:

- **Visualizações geográficas:** Mapas de calor e gráficos que mostram a distribuição geográfica de vendas, clientes, vendedores e outras métricas importantes.
- **Comparação de desempenho entre regiões:** Identificação de áreas com alto e baixo desempenho em vendas, tempo de entrega e valor do frete, permitindo ações direcionadas para otimizar os resultados em cada região.
- **Análise da performance dos vendedores por localização:** Avaliação do desempenho individual dos vendedores em diferentes cidades e estados, possibilitando a identificação de áreas com necessidade de treinamento ou realocação de recursos.
- **Identificação de preferências regionais:** Descoberta de padrões de consumo em diferentes localidades, permitindo a criação de campanhas de marketing e ofertas personalizadas para cada região.
- **Cálculo de distâncias e tempos de entrega:** Estimativa da distância média entre cliente e vendedor e do tempo médio de entrega por região, auxiliando na otimização da logística e no gerenciamento das expectativas dos clientes.

Além das análises já demonstradas pelas queries SQL, podemos explorar outras possibilidades, como:

- **Comparação do valor do frete com a média por região:** Identificar pedidos com frete acima ou abaixo da média para a cidade ou estado de destino, possibilitando a análise de outliers e a otimização da política de fretes.

- **Análise da sazonalidade das vendas por região:** Avaliar como as vendas variam ao longo do ano em diferentes localidades, permitindo a criação de promoções e campanhas de marketing sazonais.
- **Segmentação de clientes por região:** Criar grupos de clientes com base em suas características geográficas e comportamentos de compra, possibilitando a personalização da comunicação e das ofertas.

  Com a flexibilidade proporcionada pela modelagem dimensional e a riqueza das informações geográficas disponíveis, o potencial para gerar insights estratégicos e direcionar ações de negócio é imenso. A camada de visualização permitirá que esses insights sejam explorados de forma interativa e intuitiva, facilitando a tomada de decisões e impulsionando o crescimento da empresa.

Foi adicionado ao repositório atual do projeto no diretório 06 - Modelagem de Dados arquivos *.csv com as tabelas do DW. Essas tabelas serão importadas para dentro da plataforma da dadosfera.

TB__SO9P9J__DIM_CUSTOMER

TB__RM8ATH__DIM_DATA

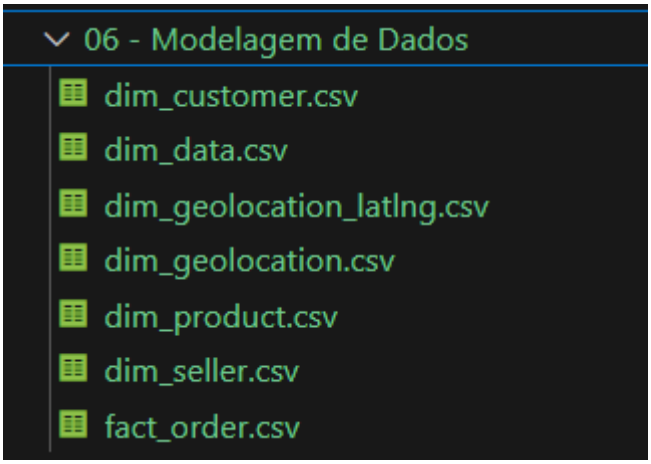
TB__IAXQ35__DIM_GEOLOCATION_LATLNG

TB__F6F8US__DIM_GEOLOCATION

TB__HKR4H4__DIM_PRODUCT

TB__8HMZ94__DIM_SELLER

TB__JJF1J5__FACT_ORDER



Documentação da Modelagem Dimensional Olist

Este README descreve a estrutura do Data Warehouse (DW) `olist_dw` e suas tabelas, construído a partir da base de dados da Olist. O objetivo principal é fornecer informações sobre cada tabela, incluindo sua finalidade, colunas e relacionamentos, para facilitar a compreensão e utilização do DW em análises e visualizações de dados.

Tabelas de Dimensão:

`dim_customer`
Finalidade: Armazena informações sobre os clientes.
Colunas:
`customer_sk`: Chave primária da tabela, gerada automaticamente (UUID).
`customer_id`: ID original do cliente na base de dados da Olist.
`customer_unique_id`: ID único do cliente.
`customer_zip_code_prefix`: CEP do cliente.
`customer_city`: Cidade do cliente.
`customer_state`: Estado do cliente.

`dim_geolocation`
Finalidade: Armazena informações geográficas sobre os CEPs, como cidade e estado.
Colunas:
`geolocation_sk`: Chave primária da tabela, gerada automaticamente (UUID).
`geolocation_zip_code_prefix`: CEP.
`geolocation_city`: Cidade.
`geolocation_state`: Estado.

`dim_geolocation_latlng`
Finalidade: Armazena informações geográficas sobre os CEPs, como latitude e longitude.
Colunas:
`geolocation_latlng_sk`: Chave primária da tabela, gerada automaticamente (UUID).
`geolocation_zip_code_prefix`: CEP.
`geolocation_lat`: Latitude.
`geolocation_lng`: Longitude.

`dim_product`
Finalidade: Armazena informações sobre os produtos.
Colunas:
`product_sk`: Chave primária da tabela, gerada automaticamente (UUID).
`product_id`: ID original do produto na base de dados da Olist.
`product_category_name`: Nome da categoria do produto.
`product_name_lenght`: Comprimento do nome do produto.
`product_description_lenght`: Comprimento da descrição do produto.
`product_photos_qty`: Quantidade de fotos do produto.
`product_weight_g`: Peso do produto em gramas.
`product_length_cm`: Comprimento do produto em centímetros.
`product_height_cm`: Altura do produto em centímetros.
`product_width_cm`: Largura do produto em centímetros.

`dim_seller`

Finalidade: Armazena informações sobre os vendedores.

Colunas:

`seller_sk`: Chave primária da tabela, gerada automaticamente (UUID).

`seller_id`: ID original do vendedor na base de dados da Olist

`seller_zip_code_prefix`: CEP do vendedor.

`seller_city`: Cidade do vendedor.

`seller_state`: Estado do vendedor.

`dim_data`

Finalidade: Armazena informações sobre datas, permitindo análises temporais.

Colunas:

`date_sk`: Chave primária da tabela, gerada automaticamente (UUID).

`data_completa`: Data completa.

`ano`: Ano.

`mes`: Mês.

`dia`: Dia.

`dia_do_ano`: Dia do ano.

`semana_do_ano`: Semana do ano.

`dia_da_semana`: Dia da semana (numérico).

`nome_dia_da_semana`: Nome do dia da semana.

`estacao`: Estação do ano.

Tabela de Fatos:

`fact_order`

Finalidade: Armazena informações sobre os pedidos, incluindo métricas de vendas, frete e datas.

Colunas:

`order_id`: ID original do pedido na base de dados da Olist

`customer_sk`: Chave estrangeira para a dimensão `dim_customer`.

`product_sk`: Chave estrangeira para a dimensão `dim_product`.

`seller_sk`: Chave estrangeira para a dimensão `dim_seller`.

`customer_geolocation_sk`: Chave estrangeira para a dimensão `dim_geolocation` (localização do cliente).

`seller_geolocation_sk`: Chave estrangeira para a dimensão `dim_geolocation` (localização do vendedor).

`shipping_limit_date`: Data limite para envio do pedido.

`price`: Preço do pedido.

`freight_value`: Valor do frete do pedido.

`order_purchase_date_sk`: Chave estrangeira para a dimensão `dim_data` (data da compra).

`order_approved_at_sk`: Chave estrangeira para a dimensão `dim_data` (data da aprovação).

`order_delivered_carrier_date_sk`: Chave estrangeira para a dimensão `dim_data` (data de entrega à transportadora).

`order_delivered_customer_date_sk`: Chave estrangeira para a dimensão `dim_data` (data de entrega ao cliente).

`order_estimated_delivery_date_sk`: Chave estrangeira para a dimensão `dim_data` (data estimada de entrega).

Relacionamentos:

`fact_order.customer_sk` -> `dim_customer.customer_sk`
`fact_order.product_sk` -> `dim_product.product_sk`
`fact_order.seller_sk` -> `dim_seller.seller_sk`
`fact_order.customer_geolocation_sk` -> `dim_geolocation.geolocation_sk`
`fact_order.seller_geolocation_sk` -> `dim_geolocation.geolocation_sk`
`fact_order.order_purchase_date_sk` -> `dim_data.date_sk`
`fact_order.order_approved_at_sk` -> `dim_data.date_sk`
`fact_order.order_delivered_carrier_date_sk` -> `dim_data.date_sk`
`fact_order.order_delivered_customer_date_sk` -> `dim_data.date_sk`
`fact_order.order_estimated_delivery_date_sk` -> `dim_data.date_sk`
`dim_geolocation.geolocation_zip_code_prefix` -> `dim_geolocation_latlng.geolocation_zip_code_prefix`

Observações:

A chave primária de cada tabela de dimensão é uma chave subsituta gerada automaticamente (UUID) para garantir a unicidade e facilitar a integração com outras fontes de dados.
A tabela de fatos `fact_order` contém as chaves estrangeiras que estabelecem os relacionamentos com as tabelas de dimensão, permitindo análises que combinam informações de diferentes dimensões.
A dimensão `dim_geolocation` foi dividida em duas tabelas para evitar problemas de chave composta em ferramentas de BI e otimizar o desempenho das consultas.
As queries SQL apresentadas anteriormente demonstram como utilizar esse modelo dimensional para responder a perguntas de negócio relevantes, tanto em termos gerais quanto com foco em análises geográficas.