Processo Seletivo – Vaga Analytics Engineer

Candidato: Thállys Lisboa Simões

Anexo essa documentação para facilitar a entrega do desafio, juntamente com todos os arquivos complementares disponíveis no link do

GitHub = https://github.com/Thallys15/Analytics-Engineer-Hotmart

Desafio Técnico - Analytics Engineer

Exercício 1 – SQL

Para este exercício, não é necessário executar nenhum código. O objetivo é escrever um SQL que considere ser a forma mais adequada de responder as perguntas abaixo, com base nas tabelas a seguir.

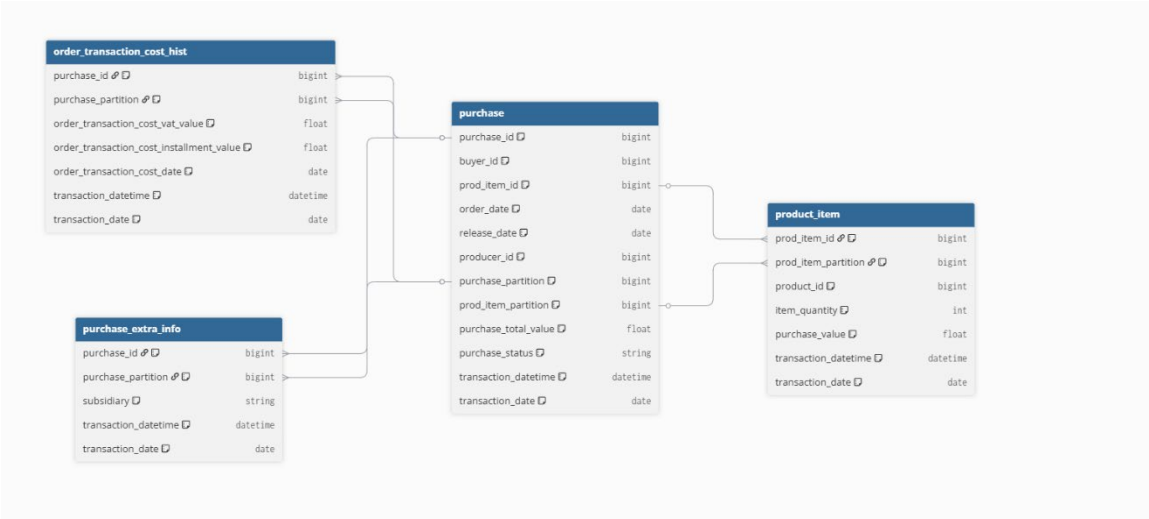
A tabela de compra possui toda a informação das transações efetuadas na plataforma. Abaixo

purchase (compra corrente)

purchase_id	buyer_id	prod_item_id	order_date	release_date	producer_id	purchase_partition	prod_item_partition
55	15947	5	2022-12-01	2022-12-01	852852	5	5
56	369798	746520	2022-12-25	2022-12-25	963963	6	0
57	147	98736	2021-07-03	2021-07-03	963963	7	6
58	986533	6565	2021-10-12	NULL	200478	8	5

product_item (item_produto corrente)

prod_item_id	product_id	item_quantity	purchase_value	prod_item_partition
1	69	5	500,00	5
5	69	120	1,00	0
98736	37	69	25,00	6
3	96	369	140,00	5



A tabela de item_produto possui os dados de quantidade de itens na transação e o valor dos itens da compra.

Respostas para o Exercício 1

Para esse exercício é importante levar em consideração que:

- o faturamento ele somente vai contar quando a compra é paga, então a ideia é filtrar pelo *release_date is not null*

- para saber os top produtores fiz um sum do *purchase_total_value* aparece lá no diagrama, por *producer_id*

Quais são os 50 maiores produtores em faturamento ($) de 2021?

```sql
select
    p.producer_id,
    sum(p.purchase_total_value) as total_revenue
where p.release_date is not null
  and extract(year from p.release_date) = 2021
group by p.producer_id
order by total_revenue desc
limit 50;
```

- *extract (year from release_date):* pra garantir que estamos pegando 2021 com base na **data de pagamento**, e não na data do pedido.

- o Sum traz a soma de toda a receita/faturamento e no final o limit 50 pra pegar o top 50.

- para calcular o top produtos, realizamos um sum no Purchase_value
- para pegar so os 2 produtos principais de cada produtos, usei row_number()

Quais são os 2 produtos que mais faturaram ($) de cada produtor?

```sql
with revenue_per_product as (
    select
        p.producer_id,
        pi.product_id,
        sum(pi.purchase_value) as product_revenue
    from purchase p
    join product_item pi
      on p.prod_item_id = pi.prod_item_id
    where p.release_date is not null
    group by p.producer_id, pi.product_id
),
ranked_products as (
    select
        producer_id,
        product_id,
        product_revenue,
        row_number() over (partition by producer_id order by product_revenue desc) as rn
    from revenue_per_product
)
select
    producer_id,
    product_id,
    product_revenue
from ranked_products
where rn <= 2
order by producer_id, product_revenue desc;
```

Aqui criei uma CTE chamada revenue_per_product pra calcular a receita total por produto de cada produtor

Depois um Join entre purchase e product_item

Essa query ela filtra apenas compras com release_date preenchida, ou seja, que tem dados e foram efetivamente realizadas

No fim agrupo por producer_id e product_id para fazer o sum do valor de cada produto.

Na segunda cte foi pra ranquear os produtos por receita dentro de cada produtos, usando a função row_number() com partition by producer_id com isso, reinicia a contagem para cada produtor.

Já na parte final pego apenas os 2 produtos mais vendidos por produtor rn <= 2 e ordena pelo producer_id e pela receita.

Exercício 2 - Modelagem e desenvolvimento

Gross Merchandising Value, ou (GMV), é o valor transacionado considerando apenas as transações cujo pagamento foi efetuado e não foi cancelado. **Nesse exercício você tem como objetivo entregar o GMV diário por subsidiária** e para isso precisará “construir” um ETL baseado nos eventos de purchase, product_item e purchase_extra_info.

purchase (eventos/cdc)

purchase (eventos/cdc)

```
purchase (eventos)
```

transaction_datetime	transaction_date	purchase_id	buyer_id	prod_item_id	order_date	release_date	producer_id
2023-01-20 22:00:00	2023-01-20	55	15947	5	2023-01-20	2023-01-20	852852
2023-01-26 00:01:00	2023-01-26	56	369798	746520	2023-01-25	NULL	963963
2023-02-05 10:00:00	2023-02-05	55	160001	5	2023-01-20	2023-01-20	852852
2023-02-26 03:00:00	2023-02-26	69	160001	18	2023-02-26	2023-02-28	96967
2023-07-15 09:00:00	2023-07-15	55	160001	5	2023-01-20	2023-03-01	852852

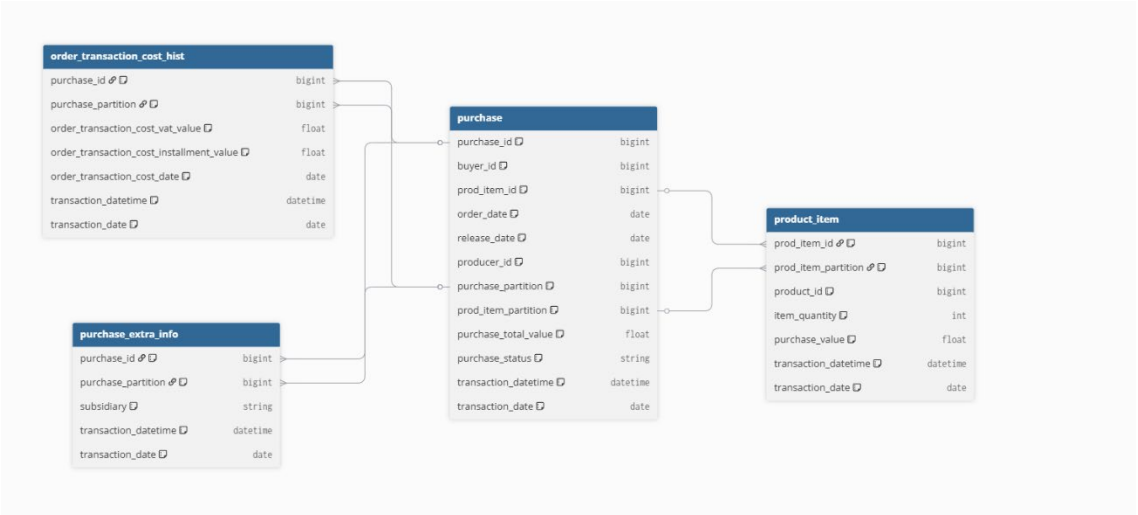
product_item (eventos/cdc)

```
product_item (eventos)
```

transaction_datetime	transaction_date	purchase_id	product_id	item_quantity	purchase_value
2023-01-20 22:02:00	2023-01-20	55	696969	10	50,00
2023-01-25 23:59:59	2023-01-25	56	808080	120	2400,00
2023-02-26 03:00:00	2023-02-26	69	373737	2	2000,00
2023-07-12 09:00:00	2023-07-12	55	696969	10	55,00

Purchase_extra_info (eventos/cdc)

transaction_datetime	transaction_date	purchase_id	subsidiary
2023-01-23 00:05:00	2023-01-23	55	nacional
2023-01-25 23:59:59	2023-01-25	56	internacional
2023-02-28 01:10:00	2023-02-28	69	nacional
2023-03-12 07:00:00	2023-03-12	69	internacional



O GMV, tecnicamente, é a soma do valor das transações, ou seja, apenas transações com "Data Liberação" preenchida, indicando que o pagamento foi efetuado.

A atualização dos dados pode ocorrer de maneira assíncrona e, em caso de falhas no envio de dados para o lake, o reenvio de eventos pode acontecer - tanto para corrigir algo hoje, quanto para mudar o passado. Porém, sempre haverá um registro equivalente para cada compra transacionada (ex: uma compra sempre terá um item de compra e uma informação extra, porém, podem não chegar no mesmo dia e hora).

Como seria a modelagem histórica e imutável de uma tabela final com o GMV acumulado do dia e separado por subsidiária? A modelagem precisa ajudar pessoas que não possuem conhecimentos sólidos em SQL.

Pré-Requisitos

- A interpretação do dado das tabelas de eventos acima descritas, faz parte da solução que precisa ser desenvolvida.
- Os dados nas tabelas acima refletem um cenário real, onde podem ocorrer inconsistências, como dados faltantes. Portanto, esperamos que a sua solução seja montada em cima dos exemplos acima fornecidos.
- Todas as tabelas são gatilhos para atualização da tabela final.
- Se uma tabela sofreu atualização e as demais não, os dados ativos das demais, deverão ser repetidos.
- A atualização ocorre em D-1.
- A modelagem precisa garantir que o passado não seja alterado, mesmo com o reprocessamento full da tabela.
- É necessário que o usuários consiga navegar entre o valor de Jan/2023 em 31/03/2023 e o valor de Jan/2023 no dia de hoje e os valores retornados pela consulta não podem ser diferentes.
- É necessário ter a rastreabilidade na granularidade diária.
- A partição da tabela pode ser o transaction_date.
- É necessário recuperar facilmente quais são os registros correntes da base histórica.
- Fica ao seu critério qual linguagem de programação utilizar: ***Preferencialmente Python, Spark ou Scala.***
- Escreva um select em cima da sua tabela para trazer a resposta para o GMV com os dados ativos/correntes de hoje.

Resolução do Exercício 2

Script do ETL - (preferencialmente em python, spark ou scala);

```python
from pyspark.sql import functions as F, Window

# 1) Leitura dos dados
df_purchase = spark.read.parquet("/raw/purchase")
df_product_item = spark.read.parquet("/raw/product_item")
df_extra = spark.read.parquet("/raw/purchase_extra_info")

#Junta todass as tabelas pelo ID da compra
df_joined = df_purchase.join(df_product_item, "purchase_id", "outer") \
                       .join(df_extra, "purchase_id", "outer")

#Marca se a compra foi paga
df_joined = df_joined.withColumn(
    "is_paid",
    F.when(F.col("release_date").isNotNull(), F.lit(True)).otherwise(F.lit(False))
)

#Marca qual é a versão mais atual por compraa
w = Window.partitionBy("purchase_id").orderBy(F.col("transaction_datetime").desc())
df_joined = df_joined.withColumn("rn", F.row_number().over(w))
df_joined = df_joined.withColumn("is_current", F.when(F.col("rn") == 1, True).otherwise(False))

#Escreve histórico
df_joined.write.mode("append").partitionBy("transaction_date").format("delta").save("/delta/fact_gmv_history")
```

Pra esse etl coloquei 3 dataframes com a Purchase trazendo informações de compra, o product_item com os detalhes dos itens comprados e o Purchase_extra_info com informações adicionais da compra.

As tabelas são unidas pelo Purchase_id e o tipo de Join é o outer pq ele garante que nenhuma compra seja perdida mesmo que falta dados na tabela

Criei uma nova coluna com o o is_paid e definindo True se a release_date tover preenchida mostrando que realmente foi pago.

Criei também uma janela pelo puchase_id e orgenando da mais recente para a mais antiga transaction_datetime, usei a row_number() pra numerar cada linha da mais recente para a mais antiga e o is_current = true pra trazer a linha mais recenete ded cada compra, com isso tenho a distinção do histórico atual x anterior.

Por fim gravei o dataframe em um Delta Lake e utilizei o append pra ir acumulando o histórico e fiz o particionamento pelo transaction_date conforme solicitado no vídeo do teste

Com isso a gente tem uma tablea com histórico do GMV, com versões e status ao longo do tempo.

Create table do dataset final - (DDL);

```
create table fact_gmv_history (
    purchase_id bigint,         -- id da transação
    product_id bigint,          -- produto (para granularidade de item)
    producer_id bigint,         -- produtor (para análise por produtor)
    subsidiary string,          -- subsidiária (nacional ou internacional)
    transaction_date date,      -- data do snapshot (partição)
    release_date date,          -- data de pagamento
    purchase_value float,       -- valor da transação
    item_quantity int,          -- quantidade do item
    is_paid boolean,            -- flag se foi pago
    is_current boolean,         -- flag se é versão mais recente
    created_at timestamp        -- data/hora da versão
)
partitioned by (transaction_date);
```

Essa tabela será o **histórico consolidado** de todas as transações, com informações sobre produtos, produtores, valores, status e datas e da pra ver o gmv por produtor, produto ou subsidiaria, filtragem de compras pagas e comparação ao loingo do tempo, e a partição dela pelo transaction_date onde cada partição corresponde a um dia de transações.

Exemplo do dataset final populado;

purchase_id	transaction_date	release_date	purchase_value	subsidiary	Observação
1	20/01/2023	20/01/2023	100.0	Nacional	Compra feita e paga no mesmo dia
1	05/02/2023	20/01/2023	80.0	Nacional	Ajuste retroativo de valor da mesma compra
2	25/01/2023	—	200.0	Internacional	Compra criada, mas ainda não paga
2	28/01/2023	28/01/2023	200.0	Internacional	Compra paga
2	10/02/2023	28/01/2023	200.0	Nacional	Alteração de metadado (subsidiária)
3	15/02/2023	15/02/2023	150.0	Internacional	Compra feita e paga no mesmo dia

O que esse dataset demonstra:

transação 1:

Valor inicial em 20/01: R$ 100

Ajuste retroativo em 05/02: R$ 80 mas com release_date de 20/01 ou seja, o GMV de jan muda se você olhar hoje.

Transação 2

Criada em 25/01 ainda não paga não entra no GMV)

Paga em 28/01 entra no GMV a partir desse dia.

Mudança de subsidiária em 10/02 não muda o passado, apenas a versão atual.

Transação 3:

Compra simples paga no mesmo dia.

O exemplo do dataset final populado foi feito justamente pra mostrar, de forma bem prática, como a modelagem pensada atende aos pontos principais do desafio.

Cada linha representa um registro do dia da transação, o que ajuda a lidar bem com mudanças ao longo do tempo.

Histórico imutável: sempre que algo muda, é criada uma nova linha com uma transaction_date diferente. Assim, o passado fica registrado e não é perdido.

Reprocessamentos sem apagar nada: por exemplo, a compra 1 não desaparece de janeiro ela só ganha uma nova versão em fevereiro.

Eventos assíncronos: a compra 2 foi criada num dia e paga depois, mostrando que as tabelas podem se atualizar em momentos diferentes.

Navegação temporal: o gmv de 31/01 e o gmv de hoje não são iguais, e isso é proposital assim dá pra comparar fechamentos históricos com a visão atual.

Evitar duplicidade: a query final usa row_number pra pegar só a última versão da transação em cada dia, evitando contar a mesma coisa mais de uma vez.

Consulta SQL, em cima do dataset final que retorne o diário por subsidiária;

```sql
with last_version_per_day as (
    select
        purchase_id,
        transaction_date,
        subsidiary,
        purchase_value,
        row_number() over (
            partition by purchase_id, transaction_date
            order by created_at desc
        ) as rn
    from fact_gmv_history
    where is_paid = true
)

select
    transaction_date,
    subsidiary,
    sum(purchase_value) as gmv
from last_version_per_day
where rn = 1
group by transaction_date, subsidiary
order by transaction_date, subsidiary;
```

a cte last_version_per_day pega apenas as versões mais recentes de cada compra por dia, usando o row_number() para ranquear

Where is_paid=true considera somente as compras pagas e a consulta final soma os valores de purchase_value das versões mais recentes e agrupei pelo transaction_date e subsidiary, com isso o GMV diário por subisidiária considera apenas a versão paga de cada compra.

Descrição sobre a tech stack que viabiliza a solução;

Na camada de ingestão e transformação, a ideia seria usar Apache Spark com Python PySpark para consolidar os eventos de purchase, product_item e purchase_extra_info. Spark é uma boa escolha porque lida muito bem com grandes volumes de dados e facilita esse tipo de join assíncrono

Já na de armazenamento, a proposta é usar Delta Lake em cima de um Data Lake Isso garante versionamento temporal, histórico imutável e particionamento por transaction_date, permitindo reprocessamentos sem alterar dados passados.

Para a camada de consumo e análise, dá pra usar SQL diretamente. Em produção, essa query poderia rodar em engines analíticas como Databricks, BigQuery ou Amazon Redshift.

Por fim, para a visualização e consumo pelo negócio, ferramentas como Power BI, Tableau ou Looker poderiam ser usadas para expor o gmv diário, permitindo comparar fechamentos históricos com a visão atual de forma simples e acessível.