

# Projeto de Engenharia e Análise de Dados: Superstore Sales



Implementação e Comparação de Performance

MySQL - Cassandra - Neo4j

Nome: Beatriz Gomes

Data: Junho 2025

# Índice

<b>1. Introdução.....</b>	<b>3</b>
<b>2. Dados e Modelo do Domínio da Aplicação.....</b>	<b>3</b>
2.1 Descrição Geral dos Dados.....	3
2.2 Modelo Geral dos Dados.....	5
2.3 Definição de Questões Analíticas Genéricas.....	6
<b>3. Modelo dos Dados para Implementação, SQL.....</b>	<b>7</b>
<b>4. Questões Analíticas para Implementação, SQL.....</b>	<b>8</b>
<b>5. Implementação da Base de Dados, SQL.....</b>	<b>11</b>
5.1 Criação das tabelas.....	12
5.2 Inserção dos dados.....	12
5.3 Implementação das Consultas.....	12
5.4 Dificuldades e Soluções.....	12
<b>6. Síntese de Resultados, SQL.....</b>	<b>13</b>
<b>7. Modelo dos Dados para Implementação, Cassandra.....</b>	<b>13</b>
7.1 Tabela Principal: superstore_sales.....	14
7.2 Criação de Tabelas Específicas.....	14
<b>8. Questões Analíticas para Implementação, Cassandra.....</b>	<b>15</b>
<b>9. Implementação da Base de Dados, Cassandra.....</b>	<b>18</b>
9.1 Estrutura do banco de dados.....	18
9.2 Preparação e Importação de Dados.....	18
9.3 Justificação da Modelagem.....	20
9.4 Dificuldades e soluções.....	20
9.5 Conclusão.....	21
<b>10. Síntese de Resultados, Cassandra.....</b>	<b>21</b>
<b>11. Modelo dos Dados para Implementação, Neo4j.....</b>	<b>22</b>
<b>12. Questões Analíticas para Implementação, Neo4j.....</b>	<b>23</b>
<b>13. Implementação da Base de Dados, Neo4j.....</b>	<b>26</b>
13.1 Estrutura do banco de dados.....	26
13.2 Visualização do gráfico.....	27
13.3 Conclusão.....	28
<b>14. Síntese de Resultados, Neo4j.....</b>	<b>28</b>
Resultados.....	28
Diferenças face ao Cassandra.....	28
Anexo 1 (SQL).....	30
Anexo 2 (Cassandra).....	34
Anexo 3 (Neo4j).....	36

## 1. Introdução

Este relatório apresenta uma análise avançada de um dataset de vendas de uma Superstore (Kaggle), focada na exploração de diferentes paradigmas de bases de dados. O objetivo principal é demonstrar como modelos Relacionais (MySQL), Colunares (Apache Cassandra) e de Grafos (Neo4j) podem ser utilizados para extrair insights sobre o comportamento do consumidor, rentabilidade de produtos e padrões geográficos de vendas.

Este projeto consistiu na implementação de uma base de dados relacional em MySQL, onde foram realizadas as seguintes etapas:

1. Preparação dos Dados: Os dados foram extraídos de uma base de dados em Excel e organizados em cinco entidades principais: Clientes, Pedidos, Itens de Pedidos, Localizações e Produtos. Essa organização permitiu uma estruturação clara e eficiente dos dados para posterior implementação.
2. Implementação do Banco de Dados: Utilizando o MySQL, foram criadas tabelas relacionais para armazenar os dados, garantindo a integridade e consistência através de chaves primárias e estrangeiras. Os dados foram carregados a partir de arquivos CSV, utilizando o comando *LOAD DATA INFILE*.
3. Análise dos Dados: Foram formuladas e implementadas cinco questões analíticas, que permitiram identificar padrões e tendências relevantes, como os clientes que fizeram pedidos sempre para a mesma cidade, os produtos mais lucrativos e os clientes que mais compraram.

A abordagem utilizada permitiu não apenas a implementação de uma base de dados relacional eficiente, mas também a extração de insights valiosos a partir dos dados disponíveis. Este relatório detalha cada uma dessas etapas, desde a descrição dos dados até à implementação das consultas, passando pela análise dos resultados obtidos.

## 2. Dados e Modelo do Domínio da Aplicação

### 2.1 Descrição Geral dos Dados

Para este projeto, os dados foram extraídos de uma base de dados em Excel e organizados em cinco entidades principais: *Cliente*, *Pedido*, *ItemPedido*, *Localização* e *Produto*.

A entidade *clientes* permite a identificação dos consumidores e a segmentação do mercado. *pedidos* armazena informações acerca dos pedidos realizados, enquanto *itens\_pedidos* destacam os produtos vendidos em cada compra. A entidade *localizacoes* possibilita análises geográficas, permitindo a identificação de padrões regionais de compra. Por fim, *produtos* contém as características dos itens comercializados.

#### Entidade: Clientes

Atributo	Significado	Valores possíveis	Exemplo
<b>customer_id</b>	Identificador único do cliente	Código alfanumérico	CG-12520
<b>customer_name</b>	Nome do cliente	Texto	Claire Gute
<b>segment</b>	Categoria do cliente	Consumer, Corporate, Home Office	Consumer

#### Entidade: Produtos

Atributo	Significado	Valores possíveis	Exemplo
<b>product_id</b>	Identificador único do produto	Código alfanumérico	FUR-BO-10001798
<b>product_name</b>	Nome do produto	Texto	Bush Somerset Collection Bookcase
<b>category</b>	Categoria do produto	Texto	Furniture
<b>sub_category</b>	Subcategoria do produto	Texto	Bookcases

#### Entidade: Localizacoes

Atributo	Significado	Valores possíveis	Exemplo
<b>location_id</b>	Identificador único da localização	Código Numérico	1
<b>postal_code</b>	Código postal da cidade	Código Numérico	42420
<b>city</b>	Cidade	Texto	Henderson
<b>state</b>	Estado	Texto	Kentucky
<b>region</b>	Região	Texto	South
<b>country</b>	País	Texto	United States

#### Entidade: Pedidos

Atributo	Significado	Valores possíveis	Exemplo
<b>order_id</b>	Identificador único do pedido	Código alfanumérico	CG-12520
<b>order_date</b>	Data do pedido	Data (DD/MM/YYYY)	Claire Gute
<b>ship_date</b>	Data de envio do pedido	Data (DD/MM/YYYY)	Consumer
<b>ship_mode</b>	Tipo de envio	Standard Class, First Class, Same Day, Second Class	Standard Class
<b>customer_id</b>	chave estrangeira - entidade clientes		
<b>location_id</b>	chave estrangeira - entidade localizacoes		

#### Entidade: itens\_pedidos

Atributo	Significado	Valores possíveis	Exemplo
<b>row_id</b>	Identificador único da linha	Valor numérico	1
<b>order_id</b>	Chave estrangeira - entidade pedido	-	
<b>product_id</b>	Chave estrangeira - entidade produto	-	
<b>sales</b>	Valor venda do produto	Valor numérico	261.96
<b>quantify</b>	Quantidade do produto vendido	Valor inteiro	2
<b>discount</b>	desconto aplicado na venda do produto	Valor numérico	0.45
<b>profit</b>	lucro obtido com a venda do produto	Valor numérico	41.9136

## 2.2 Modelo Geral dos Dados

O modelo de dados geral é apresentado na Figura 1, utilizando um Diagrama Entidade-Relacionamento (DER). Esse modelo representa o domínio da aplicação das tabelas envolvidas: *Clientes*, *Pedidos*, *Itens\_Pedidos*, *Localizacoes* e *Produtos*. Ele estabelece as relações entre as entidades, garantindo a integridade e organização das informações.

- *Clientes*: Contém informações sobre os consumidores
- *Pedidos*: Regista as compras feitas pelos clientes
- *Itens\_Pedidos*: Relaciona os produtos vendidos dentro de cada pedido.
- *Localizacoes*: Contém detalhes sobre cidades, estados e regiões.
- *Produtos*: Contém características dos itens comercializados.

O diagrama ajuda a visualizar as conexões entre as tabelas e a compreender como os dados estão organizados para análise.

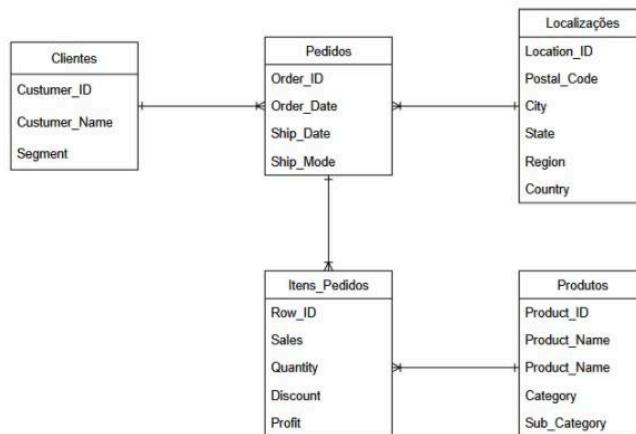


Figura 1. Modelo de dados geral

## 2.3 Definição de Questões Analíticas Genéricas

Para uma análise eficiente dos dados disponíveis, foram definidas questões analíticas que permitiram identificar padrões e tendências relevantes. As questões foram elaboradas com base na necessidade de compreender melhor o comportamento dos clientes, a rentabilidade dos produtos e a distribuição das vendas ao longo do tempo. A seguir, apresenta-se a Tabela 1 com a descrição das questões e os dados considerados para cada análise.

Questão	Dados e horizonte temporal a considerar
Quais os clientes que fizeram pedidos sempre para a mesma cidade?	<ul style="list-style-type: none"> <li><b>Dados:</b> Tabelas <a href="#">clientes</a>, <a href="#">pedidos</a> e <a href="#">localizações</a>.</li> <li><b>Análise:</b> Verificar clientes cujos pedidos estão sempre associados à mesma cidade, sem variação.</li> <li><b>Horizonte temporal:</b> todos os anos disponíveis.</li> <li><b>Anexo(s):</b> <a href="#">SQL - 1.2.1</a>, <a href="#">CAS - 2.3.1</a>, <a href="#">NEO - 3.2.5</a></li> </ul>
Qual o profit, agrupado por categorias, dos top 3 meses de um determinado ano?	<ul style="list-style-type: none"> <li><b>Dados:</b> Tabelas <a href="#">pedidos</a>, <a href="#">itens_pedidos</a> e <a href="#">produtos</a>.</li> <li><b>Análise:</b> Calcular o profit total por categoria e identificar os três meses mais lucrativos dentro de um ano específico.</li> <li><b>Horizonte temporal:</b> ano escolhido pelo usuário.</li> <li><b>Anexo(s):</b> <a href="#">SQL - 1.2.2</a>, <a href="#">CAS - 2.3.2</a>, <a href="#">NEO - 3.2.4</a>, <a href="#">G.001</a></li> </ul>
Quais são os top 3 produtos que tiveram mais de 20 vendas, mas geraram prejuízo no total?	<ul style="list-style-type: none"> <li><b>Dados:</b> Tabelas <a href="#">itens_pedidos</a> e <a href="#">produtos</a>.</li> <li><b>Análise:</b> Identificar produtos com mais de 20 unidades vendidas, mas cujo total de profit seja negativo.</li> <li><b>Horizonte temporal:</b> todos os anos disponíveis.</li> <li><b>Anexo(s):</b> <a href="#">SQL - 1.2.3</a>, <a href="#">CAS - 2.3.3</a>, <a href="#">NEO - 3.2.3</a>, <a href="#">G.002</a></li> </ul>
Quais os clientes que vivem na região sul e compraram produtos da categoria Office Supplies?	<ul style="list-style-type: none"> <li><b>Dados:</b> Tabelas <a href="#">clientes</a>, <a href="#">pedidos</a>, <a href="#">itens_pedidos</a>, <a href="#">produtos</a> e <a href="#">localizacoes</a>.</li> <li><b>Análise:</b> Filtrar clientes da região sul e verificar se compraram produtos da categoria "Office Supplies".</li> <li><b>Horizonte temporal:</b> todos os anos disponíveis.</li> <li><b>Anexo(s):</b> <a href="#">SQL - 1.2.4</a>, <a href="#">CAS - 2.3.4</a>, <a href="#">NEO - 3.2.2</a>, <a href="#">G.003</a></li> </ul>
Quais os produtos pedidos pelos 3 clientes que mais compraram (em dólares)?	<ul style="list-style-type: none"> <li><b>Dados:</b> Tabelas <a href="#">clientes</a>, <a href="#">pedidos</a>, <a href="#">itens_pedidos</a> e <a href="#">produtos</a>.</li> <li><b>Análise:</b> Identificar os três clientes com maior volume de compras em valor monetário e listar os produtos adquiridos por eles.</li> <li><b>Horizonte temporal:</b> todos os anos disponíveis.</li> <li><b>Anexo(s):</b> <a href="#">SQL - 1.2.5</a>, <a href="#">CAS - 2.3.5</a>, <a href="#">NEO - 3.2.1</a></li> </ul>

Tabela 1. Questões Analíticas

### 3. Modelo dos Dados para Implementação, SQL

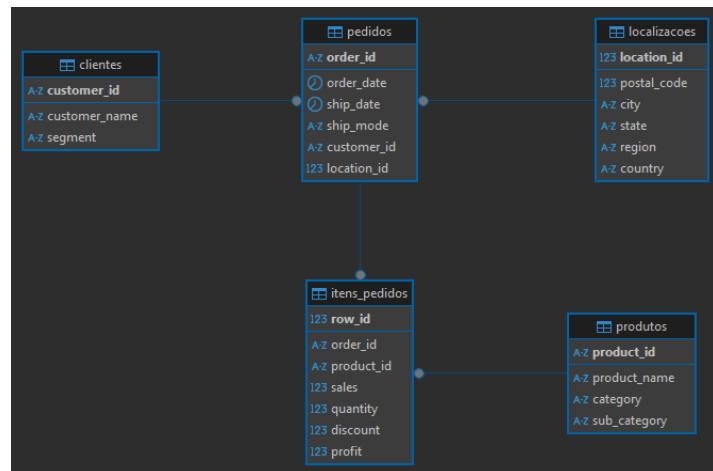


Figura 2. Modelo de dados implementado na Parte I

O modelo de dados implementado foi construído com base no modelo de domínio definido inicialmente. Durante o processo de transformação, foi necessário adaptar a estrutura para garantir uma organização eficiente dos dados, evitar redundâncias e assegurar a integridade das relações entre as tabelas.

Para converter este modelo num esquema relacional adequado para implementação na base de dados, realizamos as seguintes transformações:

#### Separação entre Pedidos e Itens\_Pedidos

No modelo de domínio, um pedido pode conter vários produtos. Para garantir que essa relação fosse corretamente representada, criamos uma tabela intermediária de *itens\_pedidos*, que liga cada pedido a um ou mais *produtos*.

Desta forma, podemos evitar a duplicação de informação na tabela *pedidos*, garantindo um modelo mais eficiente e permitindo que cada produto seja associado a vários pedidos sem necessidade de repetir dados desnecessariamente.

#### Estruturação da Informação de Localização

Inicialmente, a informação de localização poderia estar dentro da entidade, *clientes*. No entanto, para evitar dados redundantes, criamos uma tabela separada chamada *localizacoes*, que armazena os detalhes das cidades, estados, regiões e países.

Ao invés de associarmos a localização diretamente ao cliente, optamos por ligar a localização aos pedidos. Desta forma, um mesmo cliente pode fazer pedidos para diferentes localizações sem necessidade de repetir informação.

#### Definição de Chaves Primárias e Estrangeiras

Para garantir a integridade referencial dos dados, foram definidas chaves primárias para identificar cada registo de forma única e chaves estrangeiras para manter as relações entre tabelas.

As principais relações estabelecidas foram: *pedidos.customer\_id* para *clientes.customer\_id*, no qual cada pedido está associado a um cliente, *pedidos.location\_id* para *localizacoes.location\_id*, no qual cada pedido está associado a uma localização, *itens\_pedidos.order\_id* para *pedidos.order\_id*, no qual cada item pertence a um pedido específico e de *itens\_pedidos.product\_id* para *produtos.product\_id*, no qual cada item refere-se a um produto específico. Com esta estrutura, foi possível garantir a consistência dos dados e evitar problemas como dados duplicados.

### Introdução de Novos Atributos

Na fase de implementação, foram adicionados atributos adicionais para suportar funcionalidades essenciais do sistema, no qual essas alterações garantem que a base de dados suporte todas as operações necessárias para o funcionamento do sistema, como por exemplo na tabela *pedidos*: adicionados os campos *ship\_date* e *ship\_mode* para registrar detalhes sobre o envio dos pedidos e na tabela *itens\_pedidos*: incluídos os campos *sales*, *discount* e *profit* para armazenar informações financeiras associadas a cada item vendido.

## 4. Questões Analíticas para Implementação, SQL

### Questão 1: Quais os produtos pedidos pelos 3 clientes que mais compraram?

Para responder a esta questão, foi necessário calcular o total de vendas por cliente e ordenar os resultados de forma decrescente. Para tal, foram utilizadas as tabelas *clientes*, *pedidos* e *itens\_pedidos*, sendo realizada a agregação das vendas (*Sum(i.sales)*) e a ordenação (Order by).

```
WITH Top3Clientes AS (
    SELECT
        p.customer_id,
        c.customer_name,
        SUM(i.sales) AS total_sales
    FROM pedidos p
    JOIN itens_pedidos i ON p.order_id = i.order_id
    JOIN clientes c ON p.customer_id = c.customer_id
    GROUP BY p.customer_id, c.customer_name
    ORDER BY total_sales DESC
    LIMIT 3
)

SELECT
    pr.*
FROM pedidos p
JOIN itens_pedidos i ON p.order_id = i.order_id
JOIN produtos pr ON i.product_id = pr.product_id
JOIN Top3Clientes t ON p.customer_id = t.customer_id;
```

A-Z product_id	A-Z product_name	A-Z category	A-Z sub_category
TEC-MA-10003626	Hewlett-Packard Deskjet 6540 Color Inkjet Printer	Technology	Machines
TEC-MA-10002412	Cisco TelePresence System EX90 Videoconferencing Unit	Technology	Machines
OFF-PA-10001804	Xerox 195	Office Supplies	Paper
OFF-AR-10003183	Avery Fluorescent Highlighter Four-Color Set	Office Supplies	Art

Ao analisar os itens adquiridos pelos 3 clientes que mais compraram, observa-se uma predominância de produtos de alto valor unitário, como máquinas e tecnologia (ex: *Cisco TelePresence System* e *Hewlett-Packard Deskjet 6540*).

Este resultado demonstra que o faturamento destes clientes não provém da quantidade de pedidos, mas sim da aquisição de bens de capital de alto custo. Para o negócio, este insight é fundamental para implementar um atendimento personalizado para estes três perfis, garantindo a sua retenção, uma vez que a perda de um único destes clientes teria um impacto significativo

na receita anual e para priorizar a disponibilidade de itens tecnológicos de topo de gama, visto que são os produtos âncora para os clientes que geram maior liquidez à empresa.

Questão 2: Quais os clientes que vivem na região Sul e compraram produtos da categoria "Office Supplies"?

Esta questão requer a junção das tabelas *clientes*, *pedidos*, *itens\_pedidos*, *produtos* e *localizacoes*. Para identificar os clientes, filtrámos os pedidos cuja localização está na região Sul (WHERE *l.Region* = 'South') e os produtos pertencentes à categoria "Office Supplies".

```
SELECT DISTINCT c.customer_id, c.customer_name
FROM clientes c
JOIN pedidos p ON c.customer_id = p.customer_id
JOIN localizacoes l ON p.location_id = l.location_id
JOIN itens_pedidos ip ON p.order_id = ip.order_id
JOIN produtos pr ON ip.product_id = pr.product_id
WHERE l.region = 'South'
AND pr.category = 'Office Supplies';
```

A-Z customer_id	A-Z customer_name
KC-16675	Kimberly Carter
DB-13555	Dorothy Badders
SB-20290	Sean Braxton
ME-17320	Maria Etezadi
KC-16255	Karen Carlisle
CC-12685	Craig Carroll
DK-12985	Darren Koutras
PS-18970	Paul Stevenson
RD-19585	Rob Dowd
GW-14605	Giulietta Weimer
AP-10915	Arthur Prichep
AB-10105	Adrian Barton
TB-21175	Thomas Boland

A execução desta consulta mostra clientes estratégicos da região Sul, como Adrian Barton, Dorothy Dickinson. O facto de estes clientes consumirem "Office Supplies" numa região específica indica que a Superstore possui um mercado sólido em materiais de escritório no Sul dos EUA.

A extração deste segmento de clientes é vital para ações de marketing direcionado (*Targeted Marketing*) pois permite o envio de catálogos ou promoções de material de escritório com custos de logística otimizados, uma vez que já se sabe que estes clientes estão concentrados geograficamente e permite também que, ao cruzar estes dados com o volume total da região Sul, a empresa pode decidir se deve expandir o inventário de outras categorias (como Tecnologia) para estes mesmos clientes que já confiam na marca para materiais básicos.

Questão 3: Top 3 produtos que tiveram mais de 20 vendas mas geraram prejuízo no total?

Para esta análise, foi necessário calcular a quantidade total vendida e o lucro total (SUM(ip.profit)). Apenas produtos com mais de 20 unidades vendidas e lucro negativo foram considerados (HAVING SUM(ip.quantity) > 20 AND SUM(ip.profit) < 0).

```

SELECT p.product_name,
       SUM(ip.quantity) AS total_vendas,
       SUM(ip.profit) AS prejuizo
  FROM itens_pedidos ip
JOIN produtos p ON ip.product_id = p.product_id
 GROUP BY p.product_name
 HAVING SUM(ip.quantity) > 20 AND SUM(ip.profit) < 0
 ORDER BY prejuizo ASC
 LIMIT 3;

```

A-Z product_name	123 total_vendas	123 prejuizo
Chromcraft Bull-Nose Wood Oval Conference Tables & Bases	27	-2.876,11
Bush Advantage Collection Racetrack Conference Table	33	-1.934,4
GBC DocuBind P400 Electric Binding System	27	-1.878,17

Os produtos mais vendidos mas que geram prejuízo foram, por ordem decrescente, de maior prejuízo para o menor: Chromcraft Bull-nose, Bush Advantage Collection e GBC DocuBind P400.

Este insight identifica um cenário onde vender mais significa perder mais dinheiro.

Para o negócio, isto exige:

1. Revisão da Estratégia de Preços: Verificar se estes produtos estão a ser usados como "produtos isca" (loss leaders) ou se o preço de venda está simplesmente mal calculado.
2. Análise de Descontos: Avaliar se o volume de vendas é impulsionado por descontos excessivos que anulam a margem de lucro.
3. Logística: Analisar se os custos de transporte para estas unidades específicas estão a consumir a margem bruta.

#### Questão 4: Qual é o profit, agrupado por categorias, dos top 3 meses de um determinado ano?

Esta questão analisa os meses em que houve maior lucro, agrupando por categoria de produto. Foram utilizadas funções de agregação (SUM(itens\_pedidos.profit)) e extração de data (MONTH(pedidos.order\_date)).

```

SELECT
  MONTH(pedidos.order_date) AS Mes,
  SUM(CASE WHEN produtos.category = 'Furniture' THEN itens_pedidos.profit ELSE 0 END) AS Furniture,
  SUM(CASE WHEN produtos.category = 'Office Supplies' THEN itens_pedidos.profit ELSE 0 END) AS Supplies,
  SUM(CASE WHEN produtos.category = 'Technology' THEN itens_pedidos.profit ELSE 0 END) AS Technology,
  SUM(itens_pedidos.profit) AS TotalProfit
FROM itens_pedidos
JOIN pedidos ON itens_pedidos.order_id = pedidos.order_id
JOIN produtos ON itens_pedidos.product_id = produtos.product_id
WHERE YEAR(pedidos.order_date) = 2013 -- Anos [2011,2014]
GROUP BY Mes
ORDER BY TotalProfit DESC
LIMIT 3;

```

123 Mes	123 Furniture	123 Supplies	123 Technology	123 TotalProfit
12	2.093,67	7.857,56	3.385,52	13.336,75
5	1.053,63	3.861,84	7.879,35	12.794,82
3	715,59	2.483,2	9.048,62	12.247,41

Os resultados destacam que o mês de Dezembro de 2013 foi o período de maior performance, com a categoria "Office Supplies" a atingir o pico de lucro. Este padrão poderá sugerir um forte

impacto das compras de final de ano e da renovação de stocks de escritórios para o novo ano.

A compreensão desta sazonalidade permite à Superstore uma vantagem competitiva em variados aspectos:

1. Planeamento de Stock: Reforçar o inventário de "Office Supplies" logo em Outubro ou Novembro para evitar ruturas durante o pico de Dezembro.
2. Marketing Direcionado: Alocar o maior orçamento de publicidade para campanhas Business-to-Business no quarto trimestre, aproveitando a disposição natural das empresas para gastar o orçamento restante do ano.
3. Gestão de Fluxo de Caixa: Antecipar que os primeiros meses do ano podem ter lucros inferiores aos de Dezembro, permitindo uma melhor gestão das reservas financeiras.

#### Questão 5: Quais os clientes que fizeram pedidos sempre para a mesma cidade?

Para responder a esta questão, foi necessário verificar se um cliente fez pedidos apenas para uma única cidade. A abordagem envolveu a contagem distinta de cidades associadas aos pedidos de cada cliente (HAVING COUNT(DISTINCT l.city)=1).

```
SELECT c.customer_ID, c.customer_name
FROM clientes c
WHERE c.customer_id IN (
    SELECT p.customer_id
    FROM pedidos p
    JOIN localizacoes l ON p.location_id = l.location_id
    GROUP BY p.customer_id
    HAVING COUNT(DISTINCT l.city) = 1
);
```

customer_ID	customer_name
AO-10810	Anthony O'Donnell
AR-10570	Anemone Ratner
CJ-11875	Carl Jackson
HH-15010	Hilary Holden
JC-15385	Jenna Caffey
JR-15700	Jocasta Rupert
LD-16855	Lela Donovan
MG-18205	Mitch Gastineau
PH-18790	Patricia Hirasaki
RE-19405	Ricardo Emerson
RM-19750	Roland Murray
SM-20905	Susan MacKendrick
TC-21145	Theresa Coyne

A consulta identificou um grupo específico de clientes que demonstra um padrão de consumo estritamente local. Estes clientes nunca enviaram pedidos para localizações diferentes, o que os caracteriza como um segmento de alta estabilidade geográfica.

Este insight de "fidelização local" é extremamente valioso para:

- A empresa, que pode oferecer opções de levantamento em loja ou pontos de entrega fixos (*Pick-up points*), reduzindo os custos de "last-mile delivery".
- As Campanhas de publicidade física, estas tornam-se muito mais eficazes para este grupo, uma vez que a sua localização de consumo é 100% previsível.
- Frequentemente, clientes que compram sempre para a mesma morada são pequenas empresas ou escritórios locais. Identificar este padrão permite à Superstore oferecer contratos de fornecimento recorrente.

## 5. Implementação da Base de Dados, SQL

## 5.1 Criação das tabelas

A base de dados foi implementada no MySQL, utilizando tabelas relacionais para armazenar os dados. Primeiro, as tabelas foram criadas com o comando *CREATE TABLE*, definindo as colunas, tipos de dados, chaves primárias e chaves estrangeiras. Abaixo está um exemplo da criação da tabela *Clientes*, anexo 1:

```
CREATE TABLE clientes (
    customer_id VARCHAR(50) PRIMARY KEY,
    customer_name VARCHAR(50) NOT NULL,
    segment VARCHAR(50) NOT NULL
);
```

As tabelas *pedidos*, *produtos*, *localizacoes* e *itens\_pedidos* foram criadas de forma semelhante, com chaves primárias e estrangeiras para garantir a integridade dos dados. Por exemplo, a tabela *pedidos* referencia a tabela *clientes* através da chave estrangeira *customer\_id*.

## 5.2 Inserção dos dados

Os dados foram inseridos nas tabelas utilizando o comando *LOAD DATA INFILE*, que permite carregar dados diretamente de arquivos CSV. Abaixo está um exemplo de como os dados foram carregados na tabela *Clientes*, anexo 1:

```
LOAD DATA INFILE '/var/lib/mysql/csv/Cliente.csv'
INTO TABLE clientes
FIELDS TERMINATED BY ','
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Os dados das outras tabelas foram carregados de forma semelhante.

## 5.3 Implementação das Consultas

Para responder às questões analíticas, foram implementadas consultas SQL utilizando comandos como *SELECT*, *JOIN*, *GROUP BY* e *HAVING*. Abaixo está um exemplo da consulta que identifica os clientes que fizeram pedidos sempre para a mesma cidade, anexo 2.1:

```
SELECT c.customer_ID, c.customer_name
FROM clientes c
WHERE c.customer_id IN (
    SELECT p.customer_id
    FROM pedidos p
    JOIN localizacoes l ON p.location_id = l.location_id
    GROUP BY p.customer_id
    HAVING COUNT(DISTINCT l.city) = 1
);
```

As outras consultas foram implementadas de forma semelhante, com foco em agregar e filtrar os dados conforme necessário.

## 5.4 Dificuldades e Soluções

Durante a implementação, um dos desafios foi garantir a integridade dos dados ao inserir chaves estrangeiras. Por exemplo, a tabela *pedidos* referencia a tabela *clientes* através da chave estrangeira *customer\_id*. Para garantir que todos os *customer\_id* existissem na tabela *clientes*, foi necessário validar os dados antes da inserção.

Outro desafio foi o formato das datas nos arquivos CSV, que estavam no formato MM/DD/YYYY. Para resolver isso, utilizamos a função *STR\_TO\_DATE* no MySQL para converter as datas para o

formato YYYY-MM-DD durante a inserção dos dados.

## 6. Síntese de Resultados, SQL

A análise realizada no motor MySQL permitiu transformar dados brutos em indicadores de gestão. Os principais resultados estratégicos foram:

- Identificação de Clientes de Alto Valor: O cruzamento de dados permitiu isolar os clientes que mais contribuem para a faturação total, possibilitando a criação de programas de fidelização e atendimento personalizado para este segmento.
- Segmentação Geográfica e de Categoria: A extração de perfis de consumo específicos (ex: Região Sul e material de escritório) valida a eficácia das cadeias de distribuição regionais e permite campanhas de marketing mais focadas.
- Controlo de Rentabilidade: A detecção de produtos com volume de vendas elevado, mas lucro negativo, revelou falhas na estratégia de preços ou nos custos operacionais, fornecendo à gestão a base necessária para reajustes imediatos de margem.

Durante a fase de implementação, o modelo relacional demonstrou ser ideal para garantir a integridade dos dados e a flexibilidade em consultas ad-hoc.

Desafios de Engenharia superados:

- ETL e Normalização: Foi necessária a conversão de formatos de data (`STR_TO_DATE`) e a validação de chaves estrangeiras para assegurar que a base de dados fosse uma "fonte única de verdade" (*Single Source of Truth*).
- Otimização de Consultas: A utilização de cláusulas `HAVING` e múltiplas junções (`JOINS`) permitiu realizar cálculos complexos de rentabilidade que seriam impossíveis em ficheiros isolados.

## 7. Modelo dos Dados para Implementação, Cassandra

superstore_sales		profit_by_category.month		product_summary	
customer_id	text	category	text	product_id	text
order_date	timestamp	year	int	product_name	text
order_id	text	month	int	total_quantity	int
product_id	text	total_profit	decimal	total_profit	float
customer_name	text				
segment	text				
ship_mode	text				
region	text				
state	text				
city	text				
product_name	text				
category	text				
sub_category	text				
sales	decimal				
quantity	int				
discount	decimal				
profit	decimal				

customer_unique_city	
customer_id	text
num_cities	int

O modelo de dados implementado em Cassandra foi desenvolvido com base no modelo relacional utilizado em SQL, tendo sido necessário realizar uma transformação significativa para adaptá-lo ao modelo de dados orientado a colunas do Cassandra. Ao contrário do modelo relacional, que privilegia a normalização e as relações entre tabelas, o Cassandra exige uma abordagem centrada nas queries que se pretendem executar, com foco em desempenho, escalabilidade e acesso direto aos dados.

Durante esta fase, foi feita a desnormalização dos dados. Ou seja, informações anteriormente separadas em várias entidades - como clientes, produtos, pedidos, itens\_pedidos e localizações - foram integradas numa única tabela principal: *superstore\_sales*.

## 7.1 Tabela Principal: *superstore\_sales*

Esta tabela centraliza os principais atributos necessários para análise e consulta.

- Cliente: *customer\_id*, *customer\_name*, *segment*
- Produto: *product\_id*, *product\_name*, *category*, *sub\_category*
- Pedido: *order\_id*, *order\_date*, *ship\_mode*
- Localização: *city*, *state*, *region*
- Item Pedido: *sales*, *quantity*, *discount*, *profit*

A chave primária foi definida como:

```
PRIMARY KEY ((customer_id), order_date, order_id, product_id)
```

Isto permite agrupar os dados por cliente e ordená-los por data de forma eficiente, o que facilita consultas como o histórico de pedidos por cliente.

Esta tabela serviu de base para a criação das tabelas auxiliares *customer\_unique\_city* e *product\_summary*, permitindo responder eficientemente a quatro das cinco queries desenvolvidas.

## 7.2 Criação de Tabelas Específicas

De forma a suportar queries analíticas específicas e garantir tempos de resposta rápidos, foram

criadas tabelas adicionais com dados agregados:

- profit by category month: agrupa o lucro total por categoria e por mês, permitindo análises temporais. Usada para a Query 2 - "Qual o profit, agrupado por categorias, dos top 3 meses de um determinado ano?"
- customer unique city: identifica o número de cidades para as quais os clientes fizeram pedidos, útil para verificar consistência geográfica. Utilizada para a Query 1 - "Quais os clientes que fizeram pedidos sempre para a mesma cidade?".
- product summary: resume as vendas e lucros por produto, sendo essencial para identificar produtos com prejuízo e elevada quantidade vendida. Usada na Query 3 - "Quais são os top 3 produtos que tiveram mais de 20 vendas, mas geraram prejuízo no total?".

Estas tabelas foram desenhadas com base nas queries previstas, cada uma com a sua própria chave primária otimizada para leitura eficiente.

## 8. Questões Analíticas para Implementação, Cassandra

### Questão 1: Quais os produtos pedidos pelos 3 clientes que mais compraram?

Para identificar os produtos pedidos pelos 3 clientes que mais compraram, foi utilizada a tabela *superstore\_sales*. A subquery identifica os 3 clientes com maior total de vendas (Sum(sales)), agrupados por *customer\_id* e ordenados de forma decrescente. A query principal filtra os produtos comprados por esses clientes, usando a cláusula *IN* para incluir os *customer\_id* selecionados.

Para identificar os produtos comprados pelos 3 clientes que mais compraram, a consulta realiza uma subquery para selecionar os 3 clientes com maior total de vendas. Em seguida, a query principal lista os produtos distintos adquiridos por esses clientes, usando a cláusula *IN* para filtrar pelos *customer\_id* identificados.

```
SELECT DISTINCT product_id, product_name
FROM superstore_sales
WHERE customer_id IN (
    SELECT customer_id
    FROM superstore_sales
    GROUP BY customer_id
    ORDER BY SUM(sales) DESC
    LIMIT 3);
```

	A-Z product_id	A-Z product_name
1	FUR-FU-10000073	Deflect-O Glasstique Clear Desk Accessories
2	FUR-FU-10000293	Eldon Antistatic Chair Mats for Low to Medium
3	FUR-FU-10001085	3M Polarizing Light Filter Sleeves
4	FUR-FU-10001986	Dana Fluorescent Magnifying Lamp, White, 30
5	FUR-FU-10004270	Eldon Image Series Desk Accessories, Burgundy
6	OFF-AP-10002998	Holmes 99% HEPA Air Purifier
7	OFF-AR-10000588	Newell 345
8	OFF-AR-10001547	Newell 311
9	OFF-AR-10001953	Boston 1645 Deluxe Heavier-Duty Electric Pe
10	OFF-AR-10001958	Stanley Bostitch Contemporary Electric Pencil
11	OFF-AR-10003156	50 Colored Long Pencils

### Questão 2: Quais os clientes que vivem na região Sul e compraram produtos da categoria "Office Supplies"?

Para identificar os clientes, filtramos os regtos da tabela *superstore\_sales* cuja localização (*region*) corresponde à região Sul ('South') e cuja categoria de produto (*category*) é "Office Supplies".

```
SELECT DISTINCT customer_id, customer_name
FROM superstore_sales
WHERE region='South' AND category='Office Supplies';
```

	A-Z customer_id	A-Z customer_name
1	AA-10375	Allen Arnold
2	AA-10480	Andrew Allen
3	AA-10645	Anna Andreadi
4	AB-10060	Adam Bellavance
5	AB-10105	Adrian Barton
6	AB-10165	Alan Barnes
7	AB-10255	Alejandro Ballentine
8	AC-10450	Amy Cox
9	AF-10870	Art Ferguson
10	AG-10330	Alex Grayson
11	AG-10390	Allen Goldenen

### Questão 3: Top 3 produtos que tiveram mais de 20 vendas mas geraram prejuízo no total?

Para identificar os produtos com mais de 20 unidades vendidas e com lucro negativo no total, foi utilizada a tabela agregada *product\_summary*, previamente criada com base nos dados da *superstore\_sales*. Esta tabela resume, por produto, o total de vendas (Sum(quantity)) e o lucro acumulado (Sum(profit)). A query aplica os filtros *total\_quantity > 20* e *total\_profit < 0*, ordenando os resultados por *total\_profit* ascendente e limitando a 3 produtos, de forma a obter os resultados do que é pedido.

```
DROP TABLE IF EXISTS product_summary;
CREATE TABLE product_summary (
    product_id TEXT PRIMARY KEY,
    product_name TEXT,
    total_quantity INT,
    total_profit FLOAT);

INSERT INTO product_summary (product_id, product_name, total_quantity, total_profit)
SELECT product_id, product_name, SUM(quantity) AS total_quantity, SUM(profit) AS total_profit
FROM superstore_sales
GROUP BY product_id, product_name;

SELECT * FROM product_summary
WHERE total_quantity > 20 AND total_profit < 0
ORDER BY total_profit ASC
LIMIT 3;
```

	A-Z product_id	123 total_quantity	123 total_profit	A-Z product_name
1	FUR-TA-10000198	27	-2,876.11	Chromcraft Bull-Nose Wood Oval Conference Tables & Bases
2	FUR-TA-10001889	33	-1,934.4	Bush Advantage Collection Racetrack Conference Table
3	OFF-BI-10004995	27	-1,878.17	GBC DocuBind P400 Electric Binding System

### Questão 4: Qual é o profit, agrupado por categorias, dos top 3 meses de um determinado ano?

Para identificar o lucro (profit), agrupado por categorias, nos 3 meses com maior lucro de um determinado ano, foi utilizada a tabela *profit\_by\_category\_month*, criada previamente em SQL e importada para o Cassandra. Esta tabela contém o lucro total (*total\_profit*) por categoria, mês e ano. A subquery selecionou o top 3 dos meses com maior lucro total no ano de 2014, ordenando os meses por *Sum(total\_profit)*. A query principal filtra os registos desses meses e apresenta o lucro por categoria, ordenado por mês e por *total\_profit* de forma decrescente.

```

SELECT category, total_profit, month
FROM profit_by_category_month
WHERE year = 2013 AND month IN (SELECT month
                                    FROM profit_by_category_month
                                    WHERE year = 2013
                                    ORDER BY total_profit DESC
                                    LIMIT 3);

```

	category	total_profit	month
1	Furniture	715.59	3
2	Furniture	1,053.63	5
3	Furniture	2,093.67	12
4	Office Supplies	2,483.2	3
5	Office Supplies	3,861.84	5
6	Office Supplies	7,857.56	12
7	Technology	9,048.62	3
8	Technology	7,879.35	5
9	Technology	3,385.52	12

#### Questão 5: Clientes que pediram sempre para a mesma cidade?

Para identificar os clientes que fizeram sempre os seus pedidos para a mesma cidade, foi criada a tabela *customer\_unique\_city*, que armazena o número de cidades distintas para as quais cada cliente fez encomendas. A query utilizada é *num\_cities*, que pretende filtrar apenas os clientes que têm 1 cidade distinta. A query principal lista os clientes que cumprem este critério, recorrendo à cláusula *IN* para selecionar os *customer\_id* correspondentes.

```

DROP TABLE IF EXISTS customer_unique_city;
CREATE TABLE customer_unique_city (
    customer_id TEXT PRIMARY KEY,
    num_cities INT
);

INSERT INTO customer_unique_city (customer_id, num_cities)
SELECT customer_id, COUNT(DISTINCT city) AS num_cities
FROM superstore_sales
GROUP BY customer_id;

SELECT DISTINCT customer_id, customer_name
FROM superstore_sales
WHERE customer_id IN (
    SELECT customer_id
    FROM customer_unique_city
    WHERE num_cities = 1
);

```

	A-Z customer_id	A-Z customer_name
1	AO-10810	Anthony O'Donnell
2	AR-10570	Anemone Ratner
3	CJ-11875	Carl Jackson
4	HH-15010	Hilary Holden
5	JC-15385	Jenna Caffey
6	JR-15700	Jocasta Rupert
7	LD-16855	Lela Donovan
8	MG-18205	Mitch Gastineau
9	PH-18790	Patricia Hirasaki
10	RE-19405	Ricardo Emerson
11	RM-19750	Roland Murray
12	SM-20905	Susan MacKendrick
13	TC-21145	Theresa Coyne

## 9. Implementação da Base de Dados, Cassandra

### 9.1 Estrutura do banco de dados

A keyspace utilizada chama-se ss\_sales, e nela foram criadas as seguintes tabelas:

- Tabela principal (*superstore\_sales*) que contém todos os dados relevantes para a análise de vendas incluindo cliente, produto, localização, pedido e indicadores financeiros.

```
CREATE TABLE superstore_sales (
    customer_id TEXT,
    customer_name TEXT,
    segment TEXT,
    order_date TIMESTAMP,
    order_id TEXT,
    ship_mode TEXT,
    region TEXT,
    state TEXT,
    city TEXT,
    product_id TEXT,
    product_name TEXT,
    category TEXT,
    sub_category TEXT,
    sales DECIMAL,
    quantity INT,
    discount DECIMAL,
    profit DECIMAL,
    PRIMARY KEY ((customer_id), order_date, order_id, product_id)
) WITH CLUSTERING ORDER BY (order_date DESC);
```

- Tabela de apoio, que é responsável por armazenar o lucro total por categoria e mês. Essa agregação permite análises comparativas e identificação de sazonalidades.

```
CREATE TABLE profit_by_category_month (
    category TEXT,
    year INT,
    month INT,
    total_profit DECIMAL,
    PRIMARY KEY ((category, year), month)
) WITH CLUSTERING ORDER BY (month ASC);
```

### 9.2 Preparação e Importação de Dados

Objetivo: Criar um conjunto de dados desnormalizado com todas as informações detalhadas dos pedidos.

```

        SELECT
            c.customer_id,
            c.customer_name,
            REPLACE(c.segment, '\r', '') AS segment,
            DATE_FORMAT(p.order_date, '%Y-%m-%d') AS order_date,
            p.order_id,
            p.ship_mode,
            loc.region,
            loc.state,
            loc.city,
            ip.product_id,
            REPLACE(pr.product_name, '\r', '') AS product_name,
            REPLACE(pr.category, '\r', '') AS category,
            REPLACE(pr.sub_category, '\r', '') AS sub_category,
            ip.sales,
            ip.quantity,
            ip.discount,
            ip.profit
        FROM
            pedidos p
        JOIN clientes c ON p.customer_id = c.customer_id
        JOIN localizacoes loc ON p.location_id = loc.location_id
        JOIN itens_pedidos ip ON p.order_id = ip.order_id
        JOIN produtos pr ON ip.product_id = pr.product_id
        INTO OUTFILE '/var/lib/mysql/csv/temp_orders_denormalized.csv'
        FIELDS TERMINATED BY ','
        LINES TERMINATED BY '\n';

```

O que faz:

- Seleciona dados de pedidos, clientes, localização, itens dos pedidos e produtos
- Formata a data do pedido no formato YYYY-MM-DD
- Remove quebras de linha (\r, \n) de campos de texto como segment, product\_name, category, etc
- Junta várias tabelas para enriquecer os dados:
  - o clientes (dados do cliente)
  - o localizacoes (dados geográficos)
  - o itens\_pedidos (detalhes dos produtos por pedido)
  - o produtos (informações dos produtos)
- Exporta o resultado para um arquivo CSV

```

        SELECT
            pr.category,
            YEAR(p.order_date) AS year,
            MONTH(p.order_date) AS month,
            SUM(ip.profit) AS total_profit
        FROM
            pedidos p
        JOIN itens_pedidos ip ON p.order_id = ip.order_id
        JOIN produtos pr ON ip.product_id = pr.product_id
        GROUP BY pr.category, YEAR(p.order_date), MONTH(p.order_date)
        INTO OUTFILE '/var/lib/mysql/csv/temp_profit_by_category_month.csv'
        FIELDS TERMINATED BY ','
        LINES TERMINATED BY '\n';

```

Objetivo: Criar um resumo de lucro total por categoria e por mês.

O que faz:

- Seleciona a categoria do produto, o ano e o mês do pedido.
- Soma os lucros por combinação de categoria, ano, e mês.
- Agrupa os dados por categoria, ano e mês.
- Exporta o resultado para um arquivo CSV;

Depois de obtermos os dois CSVs, utilizámos os seguintes comandos no terminal para importar os dados para as respetivas tabelas no Cassandra:

```
docker exec -it bd2_cassandra cqlsh
```

*Abre uma sessão interativa do Cassandra dentro do container Docker.*

Dentro do terminal do Cassandra, acedemos ao keyspace ss\_sales:

```
USE ss_sales;
```

*Seleciona o keyspace onde estão definidas as tabelas.*

De seguida, executamos os comandos COPY para importar os dados dos ficheiros CSV para as tabelas:

```
COPY superstore_sales (customer_id, customer_name, segment, order_date, order_id,  
ship_mode, region, state, city, product_id, product_name, category, sub_category, sales,  
quantify, discount, profit)
```

```
FROM '/var/lib/cassandra/csv/temp_orders_denormalized.csv'
```

```
WITH DELIMITER = ' ; ' AND HEADER = FALSE;
```

Importa os dados desnormalizados da tabela principal a partir do ficheiro CSV.

```
COPY profit_by_category_month
```

```
FROM '/var/lib/cassandra/csv/temp_profit_by_category_month.csv'
```

```
WITH DELIMITER = ' ; ' AND HEADER = FALSE;
```

*Importa os dados agregados por categoria e mês para a tabela auxiliar.*

Estes comandos permitiram carregar de forma eficiente os dados previamente preparados para dentro das tabelas Cassandra, garantindo que a base de dados se encontra populada e pronta para responder às queries previstas.

### **9.3 Justificação da Modelagem**

A modelagem no Cassandra foi guiada pelas queries principais do projeto, na qual se tratava das consultas por cliente e período, que foram otimizadas pela chave primária composta. Também se realizou agregações por categoria e mês já pré-calculadas e armazenadas. E por último a alta performance em leitura, que foi obtida com desnormalização e modelagem voltada para a leitura.

### **9.4 Dificuldades e soluções**

Uma das principais dificuldades enfrentadas durante o projeto foi a definição da estrutura das tabelas no Cassandra, de forma a garantir que fosse possível responder eficientemente às questões analíticas propostas, tendo em conta de que não podíamos usar os “joins” dado que o Cassandra não suporta o uso de “joins”.

Tivemos de juntar numa única tabela (*superstore\_sales*) os dados que estavam distribuídos por várias entidades no SQL, além disso a chave primária no Cassandra é composta por uma chave de partição e por uma ou mais colunas de clustering, que determinam a ordenação e a forma como os dados são distribuídos.

O Cassandra não permite fazer “Group by” fora da chave de partição e para resolver este problema tivemos de criar tabelas auxiliares com dados já agregados.

## 9.5 Conclusão

A segunda parte da implementação demonstrou como uma base de dados relacional pode ser transformada e adaptada para um ambiente NoSQL orientado a colunas, como o Cassandra. Esse modelo mostrou-se eficiente para as consultas analíticas previstas, com excelente tempo de resposta e capacidade de escalar horizontalmente conforme o volume de dados aumente.

## 10. Síntese de Resultados, Cassandra

O modelo de dados no Cassandra foi implementado a partir da base de dados original, começando pela criação de duas tabelas principais: *superstore\_sales* e *profit\_by\_category\_month*. A partir da *superstore\_sales*, foram derivadas mais duas tabelas especializadas: *customer\_unique\_city* e *product\_summary*. Essa estrutura em quatro tabelas - duas principais e duas secundárias - permite otimizar consultas específicas, melhorando a eficiência sem duplicar dados desnecessariamente.

Em relação às questões analíticas que escolhemos anteriormente podemos observar que os resultados são iguais em cada uma das queries, dado que não mexemos na forma de obter os valores de cada uma das queries.

Durante a implementação do projeto em Cassandra, foram replicadas as queries originalmente desenvolvidas em SQL, de forma a comparar os resultados obtidos em ambos os ambientes. De um modo geral, as respostas obtidas nas queries em Cassandra foram equivalentes às do modelo relacional em SQL, na maioria das questões, exceto na questão 4, que não pôde ser comparada diretamente com o SQL, devido ao formato e ao pré-processado da tabela *profit\_by\_category\_month*. Neste caso, o Cassandra exige que os dados estejam previamente organizados para permitir esse tipo de consulta, enquanto no SQL seria possível fazê-lo dinamicamente com Group by, Order by e Limit.

Para terminar, ao longo deste projeto podemos verificar e analisar as diferenças entre o SQL e o Cassandra que para além de podermos usar o “Group by” no SQL em qualquer coluna e no Cassandra apenas em colunas chave, também conseguimos perceber que o “Order by” pode ser utilizado em qualquer coluna no SQL e apenas em colunas chave de clustering das tabelas definidas no Cassandra.

Enquanto que o modelo em SQL apresenta uma maior flexibilidade para consultas dinâmicas, permitindo agrupar e ordenar dados livremente, o Cassandra exige que os dados estejam organizados à partida para cada tipo de consulta, o que implica a criação de tabelas auxiliares com dados agregados, a definição cuidadosa das chaves primárias e do clustering e o planeamento prévio das queries que se pretendem executar.

## 11. Modelo dos Dados para Implementação, Neo4j

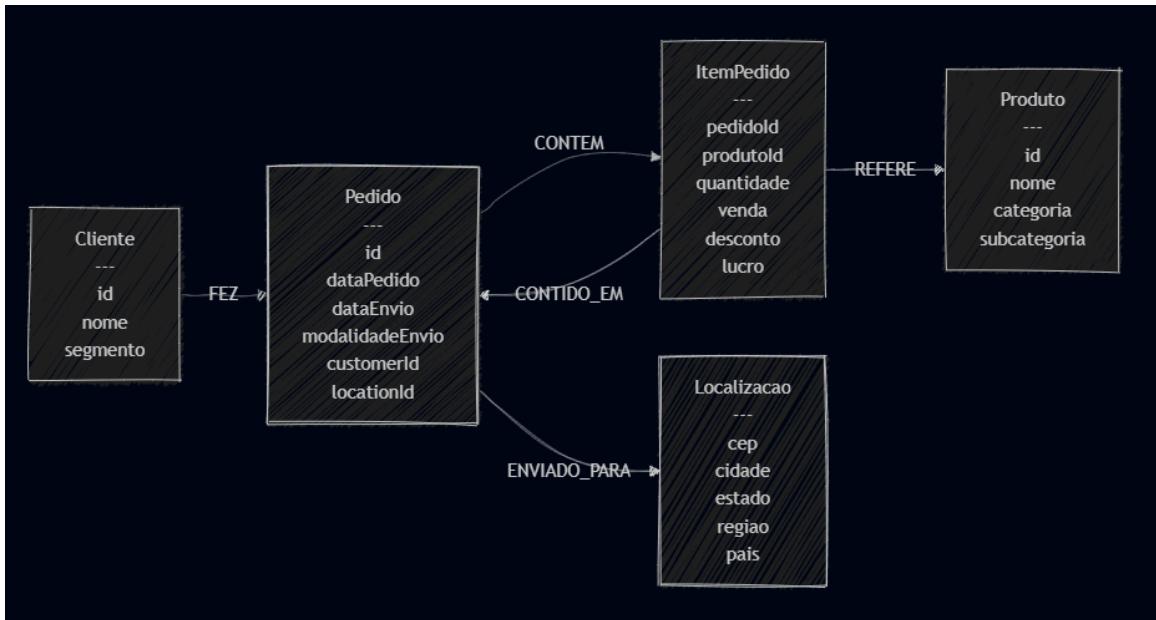


Figura 2. Representação do modelo de grafos através da linguagem Mermaid. O código-fonte está disponível no [Anexo 3.3](#).

O modelo de dados implementado nesta fase foi desenvolvido com base no modelo relacional definido inicialmente e transformado para se adequar ao paradigma de grafos, utilizado pelo sistema de base de dados Neo4j. Este tipo de base de dados é orientado à representação de entidades e das relações entre elas, o que permite uma navegação natural por meio de ligações explícitas, otimizando consultas altamente relacionais e com múltiplos saltos.

Ao contrário das bases de dados relacionais ou orientadas a colunas (como o Cassandra), onde as relações entre entidades são mantidas por chaves estrangeiras, no Neo4j essas relações são representadas de forma nativa como arestas (relacionamentos) entre nós (entidades). Essa abordagem melhora significativamente a performance em consultas que envolvem várias entidades conectadas, sendo especialmente eficaz para análises complexas de dados ligados.

### Estrutura do Grafo:

Na modelação implementada, foram criados os seguintes nós (labels):

- Cliente: Representa os clientes da loja, com os atributos id, nome e segmento
- Produto: contém as informações dos produtos, como id, nome, categoria e subcategoria
- Pedido: representa os pedidos realizados, com atributos como id, dataPedido, dataEnvio e modalidadeEnvio
- ItemPedido: representa cada linha de um pedido, com atributos quantidade, venda, desconto e lucro
- Localização: armazena os dados geográficos dos pedidos, incluindo cidade, estado, região e país

Entre estas entidades foram definidos os seguintes relacionamentos (arestas):

- (:Cliente)-[:FEZ]->(:Pedido): representa que um cliente realizou um determinado pedido
- (:Pedido)-[:CONTEM]->(:ItemPedido): representa os itens que fazem parte de um pedido
- (:ItemPedido)-[:REFERE]->(:Produto): associa cada item de um pedido ao respetivo produto
- (:Pedido)-[:ENVIADO\_PARA]->(:Localizacao): liga um pedido ao destino da entrega.
- (:ItemPedido)-[:CONTIDO\_EM]->(:Pedido): duplicação do relacionamento CONTEM para permitir consultas mais flexíveis.

Esta estrutura foi definida com o objetivo de manter a semântica do modelo de domínio, mas

adaptando-a à lógica de grafos, onde o foco está em como os dados se relacionam mais do que em como estão armazenados.

#### Criação do Grafo:

A importação dos dados foi realizada através da funcionalidade LOAD CSV, carregando os ficheiros Cliente.csv, Produto.csv, Pedido.csv, ItemPedido.csv e Localizacao.csv. Para cada uma destas entidades foram criados os nós correspondentes e, posteriormente, as relações entre eles, com base nas chaves associativas do modelo original (ex: Customer ID, Order ID, Product ID, etc.).

## 12. Questões Analíticas para Implementação, Neo4j

### Questão 1: Quais os produtos pedidos pelos 3 clientes que mais compraram?

Para resolver a query, identificaram-se os 3 clientes com maior volume total de compras, com base na soma do valor de vendas (*i.venda*). Foi utilizado `SUM(i.venda)` agrupado por cliente e ordenado de forma decrescente, retendo apenas os três primeiros. Em seguida, foi utilizado um segundo MATCH para obter os produtos comprados por esses clientes aplicando-se `DISTINCT` para eliminar duplicações, retornando apenas os produtos únicos pedidos por esses clientes.

```
MATCH (c:Cliente)-[:FEZ]->(p:Pedido)-[:CONTEN]->(i:ItemPedido)
WITH c, sum(i.venda) AS totalComprado
ORDER BY totalComprado DESC
LIMIT 3
WITH collect(c.id) AS topClientes

MATCH (c:Cliente)-[:FEZ]->(p:Pedido)-[:CONTEN]->(i:ItemPedido)-[:REFERE]->(prod:Produto)
WHERE c.id IN topClientes
RETURN DISTINCT prod.id AS product_id, prod.nome AS product_name
ORDER BY product.name
```

O	A-Z product_id	A-Z product_name
1	FUR-FU-10001085	3M Polarizing Light Filter Sleeves
2	OFF-AR-10003156	50 Colored Long Pencils
3	OFF-ST-10001558	Acco Perma 4000 Stacking Storage Drawers
4	OFF-BI-10003712	Acco Pressboard Covers with Storage Hooks, 14 7/8" x 11", Light Blue
5	OFF-LA-10003663	Avery 498
6	OFF-AR-10003183	Avery Fluorescent Highlighter Four-Color Set
7	TEC-PH-10002365	Belkin Grip Candy Sheer Case / Cover for iPhone 5 and 5S
8	OFF-AR-10001953	Boston 1645 Deluxe Heavier-Duty Electric Pencil Sharpener
9	TEC-CO-10004722	Canon imageCLASS 2200 Advanced Copier
10	TEC-MA-10002412	Cisco TelePresence System EX90 Videoconferencing Unit
11	FUR-FU-10001986	Dana Fluorescent Magnifying Lamp, White, 36"
12	FUR-FU-10000073	Deflect-O Glasstique Clear Desk Accessories
13	FUR-FU-10000293	Eldon Antistatic Chair Mats for Low to Medium Pile Carpets
14	FUR-FU-10004270	Eldon Image Series Desk Accessories, Burgundy
15	TEC-AC-10003198	Enermax Acrylux Wireless Keyboard
16	OFF-ST-10001469	Fellowes Bankers Box Recycled Super Stor/Drawer
17	OFF-BI-10004519	GBC DocuBind P100 Manual Binding Machine

### Questão 2: Quais os clientes que vivem na região Sul e compraram produtos da categoria "Office Supplies"?

Para resolver a query, foi utilizada uma consulta que liga as entidades Cliente, Pedido, ItemPedido, Produto e Localizacao. O objetivo é identificar clientes que tenham feito compras de produtos da categoria "Office Supplies" e cujos pedidos tenham sido enviados para a região Sul. A consulta percorre o seguinte caminho no grafo:

(:Cliente)-[:FEZ]->(:Pedido)-[:CONTEN]->(:ItemPedido)-[:REFERE]->(:Produto) e  
(:Pedido)-[:ENVIADO\_PARA]->(:Localizacao). Foram aplicadas depois as condições Localizações com regiao = 'South' e Produtos com categoria = 'Office Supplies' para filtrar os resultados.

```

MATCH (c:Cliente)-[:FEZ]->(p:Pedido)-[:CONTEM]->(i:ItemPedido)-[:REFERE]->(prod:Produto),
      (p)-[:ENVIADO_PARA]->(l:Localizacao)
WHERE l.regiao = 'South' AND prod.categoria = 'Office Supplies'
RETURN DISTINCT c.nome AS Cliente, l.regiao AS Regiao, prod.categoria AS Categoria

```

	A-Z Cliente	A-Z Regiao	A-Z Categoria
1	Thomas Boland	South	Office Supplies
2	Rob Dowd	South	Office Supplies
3	Arthur Prichep	South	Office Supplies
4	Alan Hwang	South	Office Supplies
5	Kimberly Carter	South	Office Supplies
6	Craig Carroll	South	Office Supplies
7	Adrian Barton	South	Office Supplies
8	Darren Koutras	South	Office Supplies
9	Sean Braxton	South	Office Supplies
10	Dorothy Badders	South	Office Supplies
11	Giulietta Weimer	South	Office Supplies
12	Paul Stevenson	South	Office Supplies
13	Maria Etezadi	South	Office Supplies
14	Karen Carlisle	South	Office Supplies
15	Darren Powers	South	Office Supplies
16	Sean O'Donnell	South	Office Supplies
17	Michael Grace	South	Office Supplies

#### Questão 3: Top 3 produtos que tiveram mais de 20 vendas mas geraram prejuízo no total?

Para responder a esta questão, foi analisada a relação entre os itens de pedido e os produtos (ItemPedido → Produto). Através da agregação dos dados, calcularam-se a quantidade total vendida e o lucro total por produto. Foram filtrados apenas os produtos que tiveram mais de 20 unidades vendidas e apresentaram lucro total negativo.

```

MATCH (i:ItemPedido)-[:REFERE]->(p:Produto)
WITH p,
     sum(i.quantidade) AS totalQuantidade,
     sum(i.lucro) AS totalLucro
WHERE totalQuantidade > 20 AND totalLucro < 0
RETURN p.nome AS produto, totalQuantidade, totalLucro
ORDER BY totalLucro ASC
LIMIT 3

```

	A-Z produto	123 totalQuantidade	123 totalLucro
1	Chromcraft Bull-Nose Wood Oval Conference Tables & Bases	27	-2,876.1156
2	Bush Advantage Collection Racetrack Conference Table	33	-1,934.3976
3	GBC DocuBind P400 Electric Binding System	27	-1,878.1662

#### Questão 4: Qual é o profit, agrupado por categorias, dos top 3 meses de um determinado ano?

Esta análise foi dividida em duas fases:

1. Identificar os 3 meses com maior lucro no ano escolhido (2012)

Primeiro, a data do pedido (dataPedido) foi dividida para extrair o mês e o ano, considerando diferentes formatos de data. Em seguida, foram somados os lucros (lucro) de cada mês e selecionados os 3 meses com maior lucro total.

2. Calcular o lucro por categoria nesses 3 meses

Depois, com os meses identificados, foi calculado o lucro total por categoria de produto, apenas para os pedidos feitos nesses meses e no ano de 2012. O objetivo foi perceber quais categorias contribuíram mais para o lucro nesses períodos.

```

MATCH (i:ItemPedido)-[:CONTIDO_EM]->(p:Pedido)
WHERE p.dataPedido IS NOT NULL
WITH i.lucro AS lucro,
    split(p.dataPedido, "/") AS partesData,
    i, p
WITH
CASE
    WHEN toInteger(partesData[0]) <= 13 THEN toInteger(partesData[0])
    ELSE toInteger(partesData[1])
END AS mes,
CASE
    WHEN toInteger(partesData[2]) >= 1000 THEN toInteger(partesData[2])
    ELSE toInteger(partesData[2]) + 2000
END AS ano,
lucro,
i
WHERE ano = 2013
WITH mes, SUM(lucro) AS lucroTotal
ORDER BY lucroTotal DESC
LIMIT 3
WITH collect(mes) AS topMeses

MATCH (i:ItemPedido)-[:CONTIDO_EM]->(p:Pedido), (i)-[:REFERE]->(prod:Produto)
WHERE p.dataPedido IS NOT NULL
WITH i.lucro AS lucro,
    split(p.dataPedido, "/") AS partesData,
    prod.categoria AS categoria,
    topMeses
WITH
CASE
    WHEN toInteger(partesData[0]) <= 13 THEN toInteger(partesData[0])
    ELSE toInteger(partesData[1])
END AS mes,
CASE
    WHEN toInteger(partesData[2]) >= 1000 THEN toInteger(partesData[2])
    ELSE toInteger(partesData[2]) + 2000
END AS ano,
categoria,
lucro,
topMeses
WHERE ano = 2013 AND mes IN topMeses
RETURN mes, categoria, SUM(lucro) AS lucroTotal
ORDER BY mes, categoria

```

	mes	categoria	lucroTotal
1	3	Furniture	715.6083
2	3	Office Supplies	2,483.2364
3	3	Technology	9,048.601
4	5	Furniture	1,053.5969
5	5	Office Supplies	3,861.8828
6	5	Technology	7,879.2944
7	12	Furniture	2,093.6403
8	12	Office Supplies	7,857.5392
9	12	Technology	3,385.5363

#### Questão 5: Quais os clientes que pediram sempre para a mesma cidade?

Esta consulta tem como objectivo identificar os clientes que enviaram todos os seus pedidos para uma única cidade.

Para isso, a análise percorre a relação entre Cliente, Pedido e Localizacao, recolhendo todas as cidades distintas para onde cada cliente fez pedidos. Em seguida, filtra-se o conjunto de clientes cuja lista de cidades tem apenas um elemento, usando size(cidades) = 1.

```

MATCH (c:Cliente)-[:FEZ]->(:Pedido)-[:ENVIADO_PARA]->(l:Localizacao)
WITH c, collect(DISTINCT l.cidade) AS cidades
WHERE size(cidades) = 1
RETURN c.id AS clienteId, c.nome AS nomeCliente

```

	clientID	nomeCliente
1	CJ-11875	Carl Jackson
2	SM-20905	Susan MacKendrick
3	AO-10810	Anthony O'Donnell
4	HH-15010	Hilary Holden
5	TC-21145	Theresa Coyne
6	AR-10570	Anemone Ratner
7	JR-15700	Jocasta Rupert
8	MG-18205	Mitch Gastineau
9	PH-18790	Patricia Hirasaki
10	LD-16855	Lela Donovan
11	RM-19750	Roland Murray
12	JC-15385	Jenna Caffey
13	RE-19405	Ricardo Emerson

## 13. Implementação da Base de Dados, Neo4j

### 13.1 Estrutura do banco de dados

Labels, que contém todos os dados relevantes para a análise de vendas incluindo cliente, produto, localização, pedido e item pedido.

#### Clientes:

```
LOAD CSV WITH HEADERS FROM 'file:///Cliente.csv' AS row
FIELDTERMINATOR ';'
WITH row WHERE row.`Customer ID` IS NOT NULL
MERGE (c:Cliente {id: row.`Customer ID`})
SET c.nome = row.`Customer Name`,
    c.segmento = row.Segment
```

#### Item Pedido:

```
LOAD CSV WITH HEADERS FROM 'file:///ItemPedido.csv' AS row
FIELDTERMINATOR ','
WITH row WHERE row.`Row ID` IS NOT NULL
MERGE (i:ItemPedido {id: row.`Row ID`})
SET i_pedidoId = row.`Order ID`,
    i_produtoId = row.`Product ID`,
    i_venda = toFloat(row.Sales),
    i_quantidade = toInteger(row.Quantity),
    i_desconto = toFloat(row.Discount),
    i_lucro = toFloat(row.Profit)
```

#### Localização:

```
LOAD CSV WITH HEADERS FROM 'file:///Localizacao.csv' AS row
FIELDTERMINATOR ','
WITH row WHERE row.`Location ID` IS NOT NULL
MERGE (l:Localizacao {id: row.`Location ID`})
SET l.cep = row.`Postal Code`,
    l.cidade = row.City,
    l.estado = row.State,
    l.regiao = row.Region,
    l.pais = row.Country
```

#### Pedido:

```

LOAD CSV WITH HEADERS FROM 'file:///Pedido.csv' AS row
FIELDTERMINATOR ';'
WITH row WHERE row.`Order ID` IS NOT NULL
MERGE (p:Pedido {id: row.`Order ID`})
SET p.dataPedido = row.`Order Date`,
    p.dataEnvio = row.`Ship Date`,
    p.modalidadeEnvio = row.`Ship Mode`,
    p.customerId = row.`Customer ID`,
    p.locationId = row.`Location ID`

```

### Produto:

```

LOAD CSV WITH HEADERS FROM 'file:///Produto.csv' AS row
FIELDTERMINATOR ';'
WITH row WHERE row.`Product ID` IS NOT NULL
MERGE (p:Produto {id: row.`Product ID`})
SET p.nome = row.`Product Name`,
    p.categoria = row.Category,
    p.subcategoria = row.`Sub-Category`

```

Relacionamentos, que são responsáveis por conectar as entidades entre si para respondermos às nossas queries.

//Cliente FEZ Pedido MATCH (c:Cliente), (p:Pedido) WHERE c.id = p.customerId MERGE (c)-[:FEZ]->(p)	//Pedido CONTEM ItemPedido MATCH (i:ItemPedido), (p:Pedido) WHERE i.itemId = p.id MERGE (p)-[:CONTENDEM]->(i)	//ItemPedido CONTIDO_EM Pedido MATCH (i:ItemPedido), (p:Pedido) WHERE i.itemId = p.id MERGE (i)-[:CONTIDO_EM]->(p)
//ItemPedido REFERE Produto MATCH (i:ItemPedido), (prod:Produto) WHERE i.itemId = prod.id MERGE (i)-[:REFERE]->(prod)	//Pedido ENVIADO_PARA Localização MATCH (p:Pedido), (l:Localizacao) WHERE p.locationId = l.id MERGE (p)-[:ENVIADO_PARA]->(l)	

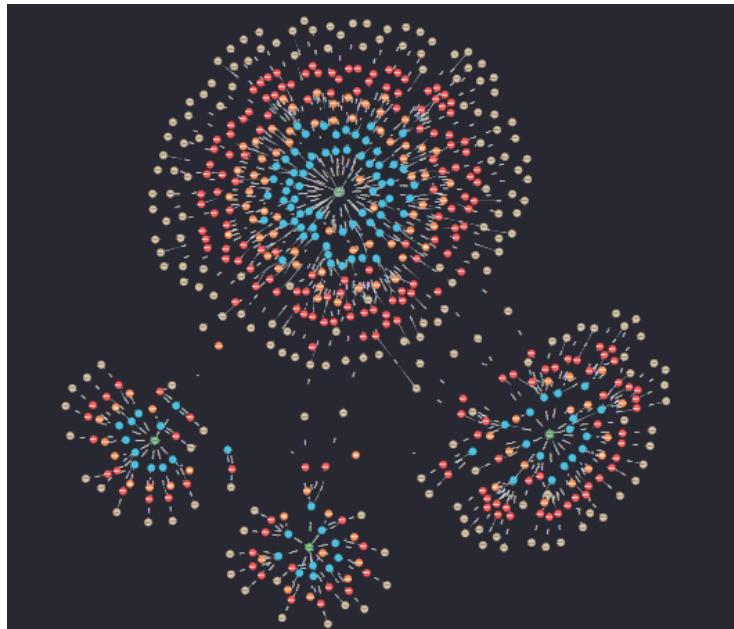
## 13.2 Visualização do gráfico

A visualização geral do grafo foi obtida com este código:

```

MATCH (c:Cliente)-[:FEZ]->(p:Pedido)-[:CONTENDEM]->(i:ItemPedido)-[:REFERE]->(prod:Produto),
      (p)-[:ENVIADO_PARA]->(l:Localizacao)
RETURN c, p, i, prod, l
LIMIT 200

```



### **13.3 Conclusão**

A utilização do Neo4j permitiu construir uma base de dados otimizada para análise de ligações complexas, com excelente desempenho em queries de exploração relacional. Este modelo provou ser eficaz na resposta a questões de negócio sobre padrões de compra, localização geográfica e identificação de produtos críticos.

## **14. Síntese de Resultados, Neo4j**

Nesta fase do projeto, implementámos as mesmas questões analíticas num modelo de grafos usando a base de dados Neo4j, permitindo uma comparação direta com os modelos previamente implementados: o modelo relacional (SQL) e o modelo baseado em colunas (Cassandra). O objetivo foi avaliar as diferenças em termos de modelação, complexidade de implementação, desempenho e expressividade das consultas.

### Resultados

Os resultados obtidos com Neo4j foram consistentes com os obtidos nas versões em SQL e Cassandra. Em todos os casos, foi possível responder às cinco questões propostas, e os dados extraídos mantiveram-se equivalentes. A grande diferença prende-se com o grau de esforço necessário para alcançar esses mesmos resultados, bem como com a forma como a informação foi expressa e consultada.

### Diferenças face ao Cassandra

Ao compararmos a implementação em Neo4j com a de Cassandra, destacaram-se várias diferenças relevantes:

#### **1. Modelação de dados:**

Em Cassandra, a estrutura das queries determinava desde o início a forma da base de dados. Fomos obrigados a desnormalizar e criar tabelas específicas para cada questão, incluindo pré-computações. A modelação foi condicionada por restrições na linguagem de consulta e pelo facto de Cassandra não permitir joins nem filtros dinâmicos eficientes.

Em contraste, a modelação em Neo4j foi mais intuitiva. O modelo relacional foi facilmente transformado em nós e relações explícitas. Essa estrutura permitiu maior flexibilidade, refletindo de forma natural as relações semânticas entre entidades.

#### **2. Linguagem de consulta:**

A linguagem Cypher, embora diferente do SQL, revelou-se expressiva e adequada para consultas complexas com múltiplas condições e relacionamentos. Ao contrário de Cassandra, não foi necessário planear previamente o “formato” da resposta - foi possível navegar pelas relações dinamicamente.

Por exemplo, consultas que no Cassandra exigiram tabelas auxiliares (como a identificação dos clientes que pediram sempre para a mesma cidade, ou produtos com prejuízo e mais de 20 vendas) foram resolvidas diretamente em Cypher com agregações simples e sem duplicação de dados.

#### **3. Desempenho:**

Observámos que os tempos de execução das consultas em Neo4j foram inferiores aos observados em Cassandra, mesmo com um volume de dados reduzido. Isto poderá ser explicado pela ausência de desnortinalização e pelo facto de a estrutura de grafos permitir percursos diretos entre nós. Contudo, importa notar que Cassandra está desenhado para cenários de Big Data com centenas de milhares de inserções por segundo, o que não se aplicava ao nosso caso. Ou seja, as suas vantagens não foram evidentes neste projeto.

## Anexo 1 (SQL)

### Anexo 1.1 - Criação e inserção de dados nas Tabelas

```
USE bd2;

-- Drop de tabelas se já existirem
DROP TABLE IF EXISTS itens_pedidos, pedidos, localizacoes, clientes, produtos;

-- Criação das tabelas
CREATE TABLE localizacoes (
    Location_id INT NOT NULL,
    postal_code INT NOT NULL,
    city VARCHAR(50) NOT NULL,
    state VARCHAR(50) NOT NULL,
    region VARCHAR(50) NOT NULL,
    country VARCHAR(50) NOT NULL,
    PRIMARY KEY (Location_id)
);

CREATE TABLE clientes (
    customer_id VARCHAR(50) PRIMARY KEY,
    customer_name VARCHAR(50) NOT NULL,
    segment VARCHAR(50) NOT NULL
);

CREATE TABLE produtos (
    product_id VARCHAR(20) PRIMARY KEY,
    product_name VARCHAR(150) NOT NULL,
    category VARCHAR(50) NOT NULL,
    sub_category VARCHAR(50) NOT NULL
);

CREATE TABLE pedidos (
    order_id VARCHAR(15) NOT NULL,
    order_date DATE NOT NULL,
    ship_date DATE NOT NULL,
    ship_mode VARCHAR(20) NOT NULL,
    customer_id VARCHAR(10) NOT NULL,
    Location_id INT NOT NULL,
    PRIMARY KEY (order_id),
    FOREIGN KEY (customer_id) REFERENCES clientes(customer_id),
    FOREIGN KEY (Location_id) REFERENCES Localizacoes(Location_id)
);

CREATE TABLE itens_pedidos (
    row_id INT NOT NULL,
    order_id VARCHAR(15) NOT NULL,
    product_id VARCHAR(20) NOT NULL,
    sales DECIMAL(10, 2) NOT NULL,
    quantity INT NOT NULL,
    discount DECIMAL(3, 2) NOT NULL,
    profit DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (row_id),
    FOREIGN KEY (order_id) REFERENCES pedidos(order_id),
    FOREIGN KEY (product_id) REFERENCES produtos(product_id)
);

-- Inserção dos dados nas tabelas
LOAD DATA INFILE '/var/Lib/mysql/csv/Localizacao.csv'
INTO TABLE Localizacoes
FIELDS TERMINATED BY ';'
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

LOAD DATA INFILE '/var/Lib/mysql/csv/Cliente.csv'
INTO TABLE clientes
FIELDS TERMINATED BY ';'
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

LOAD DATA INFILE '/var/Lib/mysql/csv/Produto.csv'
INTO TABLE produtos
```

```

FIELDS TERMINATED BY ';'
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

LOAD DATA INFILE '/var/Lib/mysql/csv/Pedido.csv'
INTO TABLE pedidos
FIELDS TERMINATED BY ';'
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(order_id, @order_date, @ship_date, ship_mode, customer_id, location_id)
SET
    order_date = STR_TO_DATE(@order_date, '%m/%d/%Y'),
    ship_date = STR_TO_DATE(@ship_date, '%m/%d/%Y');

LOAD DATA INFILE '/var/Lib/mysql/csv/ItemPedido.csv'
INTO TABLE itens_pedidos
FIELDS TERMINATED BY ';'
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```

#### Anexo 1.2 - Queries:

##### 1.2.1) Quais os clientes que fizeram pedidos sempre para a mesma cidade?

```

USE bd2;

SELECT c.customer_ID, c.customer_name
FROM clientes c
WHERE c.customer_id IN (
    SELECT p.customer_id
    FROM pedidos p
    JOIN localizacoes l ON p.location_id = l.location_id
    GROUP BY p.customer_id
    HAVING COUNT(DISTINCT l.city) = 1
);

```

##### 1.2.2) Qual o profit, agrupado por categorias, dos top 3 meses de um determinado ano?

```

USE bd2;

SELECT
    MONTH(pedidos.order_date) AS Mes,
    SUM(CASE WHEN produtos.category = 'Furniture' THEN itens_pedidos.profit ELSE 0 END) AS Furniture,
    SUM(CASE WHEN produtos.category = 'Office Supplies' THEN itens_pedidos.profit ELSE 0 END) AS Supplies,
    SUM(CASE WHEN produtos.category = 'Technology' THEN itens_pedidos.profit ELSE 0 END) AS Technology,
    SUM(itens_pedidos.profit) AS TotalProfit
FROM itens_pedidos
JOIN pedidos ON itens_pedidos.order_id = pedidos.order_id
JOIN produtos ON itens_pedidos.product_id = produtos.product_id
WHERE YEAR(pedidos.order_date) = 2013 -- Anos [2011, 2014]
GROUP BY Mes
ORDER BY TotalProfit DESC
LIMIT 3;

```

##### 1.2.3) Quais são os top 3 produtos que tiveram mais de 20 vendas, mas geraram prejuízo no total?

```

USE bd2;

SELECT p.product_name , SUM(ip.quantity) AS total_vendas,
       SUM(ip.profit) AS prejuizo
  FROM itens_pedidos ip
 JOIN produtos p ON ip.product_id = p.product_id
 GROUP BY p.product_name
 HAVING SUM(ip.quantity) > 20 AND SUM(ip.profit) < 0
        ORDER BY prejuizo ASC
 LIMIT 3;

```

1.2.4) Quais os clientes que vivem na região sul e compraram produtos da categoria Office Supplies?

```

USE bd2;

SELECT DISTINCT c.customer_id, c.customer_name
  FROM clientes c
 JOIN pedidos p ON c.customer_id = p.customer_id
 JOIN localizacoes l ON p.location_id = l.location_id
 JOIN itens_pedidos ip ON p.order_id = ip.order_id
 JOIN produtos pr ON ip.product_id = pr.product_id
 WHERE l.region = 'South'
 AND pr.category = 'Office Supplies';

```

1.2.5) Quais os produtos pedidos pelos 3 clientes que mais compraram (em dólares)?

```

USE bd2;

WITH Top3Clientes AS (
  SELECT
    p.customer_id,
    c.customer_name,
    SUM(i.sales) AS total_sales
   FROM pedidos p
  JOIN itens_pedidos i ON p.order_id = i.order_id
  JOIN clientes c ON p.customer_id = c.customer_id
 GROUP BY p.customer_id, c.customer_name
 ORDER BY total_sales DESC
 LIMIT 3)

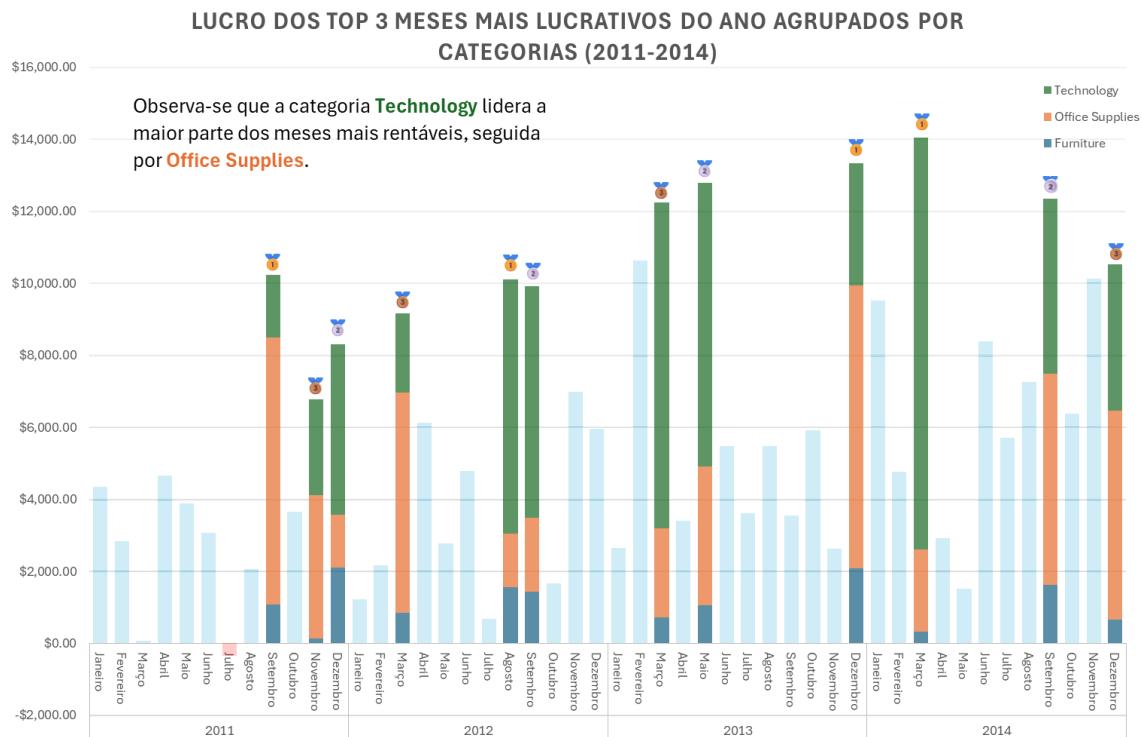
  SELECT
    pr.*
   FROM pedidos p
  JOIN itens_pedidos i ON p.order_id = i.order_id
  JOIN produtos pr ON i.product_id = pr.product_id
  JOIN Top3Clientes t ON p.customer_id = t.customer_id;

```

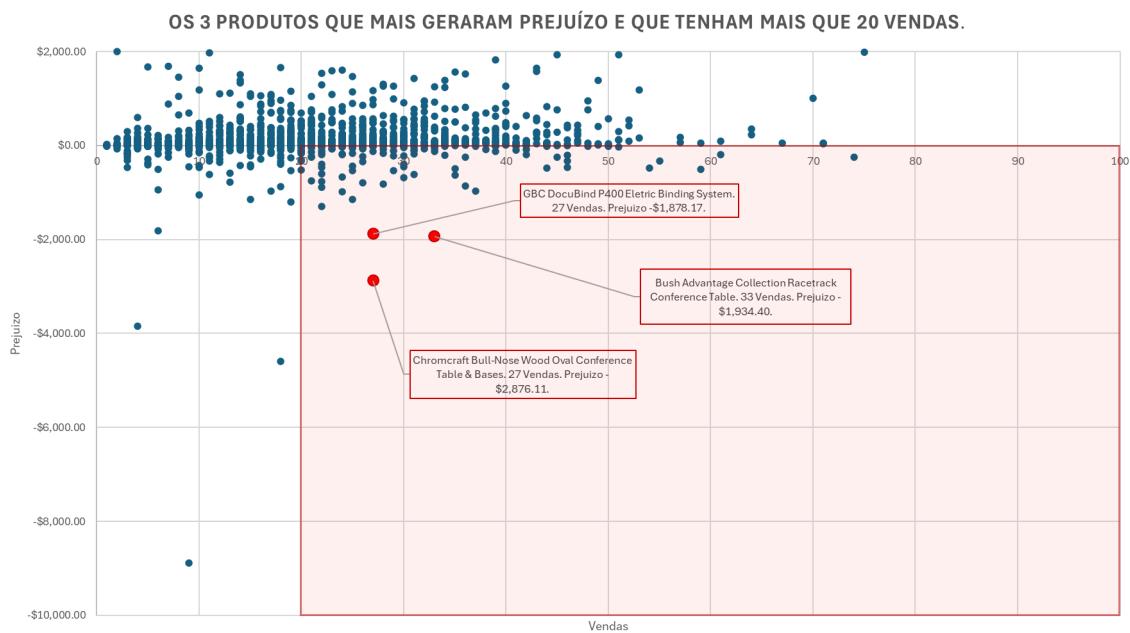
Anexo 1.3 - Gráficos:

G.001) O gráfico a seguir apresenta o lucro mensal dos três meses mais lucrativos de cada ano,

agrupados pelas categorias Technology, Office Supplies e Furniture, no período de 2011 a 2014. Os meses de maior faturamento foram destacados com medalhas, evidenciando os períodos de pico de lucratividade. Nota-se que a categoria Technology frequentemente apresenta os maiores valores de lucro, seguida por Office Supplies e Furniture.

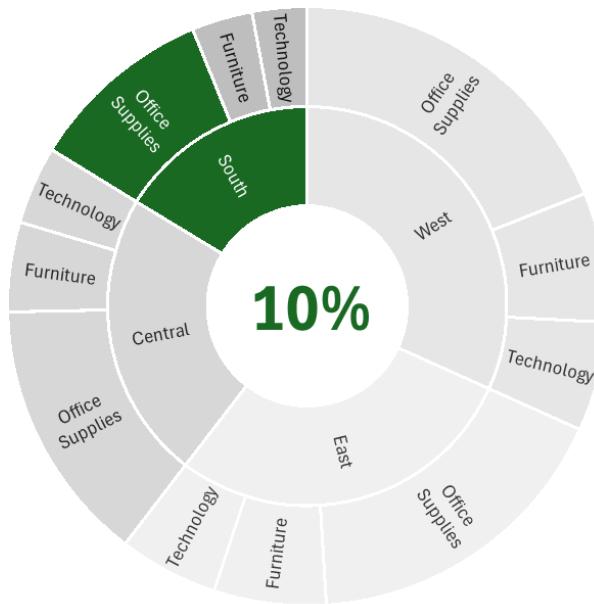


G.002) Este gráfico de dispersão apresenta a relação entre o número de vendas (eixo X) e o prejuízo acumulado (eixo Y). Dentro do retângulo vermelho estão todos os produtos que têm prejuízo e mais de 20 vendas. Sendo o Top 3 os produtos com mais prejuízo devidamente indicado.



G.003) Este gráfico 'sunburst' mostra a quantidade de clientes que vivem na região sul e que pedem produtos da categoria 'Office Supplies'.

Existem **1620** Clientes que vivem na região 'South' sendo que **995** deles compraram produtos da categoria 'Office Supplies'.



## Anexo 2 (Cassandra)

### Anexo 2.1 - Criação da tabela Cassandra

```
CREATE KEYSPACE SS_Sales WITH REPLICATION = {'class':'SimpleStrategy', 'replication_factor' : 1};

USE SS_Sales;

CREATE TABLE superstore_sales (
    customer_id TEXT,
    customer_name TEXT,
    segment TEXT,
    order_date TIMESTAMP,
    order_id TEXT,
    ship_mode TEXT,
    region TEXT,
    state TEXT,
    city TEXT,
    product_id TEXT,
    product_name TEXT,
    category TEXT,
    sub_category TEXT,
    sales DECIMAL,
    quantity INT,
    discount DECIMAL,
    profit DECIMAL,
    PRIMARY KEY ((customer_id), order_date, order_id, product_id)
) WITH CLUSTERING ORDER BY (order_date DESC);

CREATE TABLE profit_by_category_month (
    category TEXT,
    year INT,
    month INT,
    total_profit DECIMAL,
    PRIMARY KEY ((category, year), month)
) WITH CLUSTERING ORDER BY (month ASC);
```

## Anexo 2.2 - Terminal Bash

```
docker exec -it bd2_cassandra cqlsh
```

```
use ss_sales;
```

```
COPY superstore_sales (customer_id, customer_name, segment, order_date, order_id, ship_mode, region, state, city, product_id, product_name, category, sub_category, sales, quantity, discount, profit) FROM '/var/lib/cassandra/csv/temp_orders_denormalized.csv' WITH DELIMITER = ';' AND HEADER = FALSE;
```

```
COPY profit_by_category_month FROM '/var/lib/cassandra/csv/temp_profit_by_category_month.csv' WITH DELIMITER = ';' AND HEADER = FALSE;
```

## Anexo 2.3 - Queries Cassandra

### 2.3.1) Quais os clientes que fizeram pedidos sempre para a mesma cidade?

```
DROP TABLE IF EXISTS customer_unique_city;
CREATE TABLE customer_unique_city (
    customer_id TEXT PRIMARY KEY,
    num_cities INT
);

INSERT INTO customer_unique_city (customer_id, num_cities)
SELECT customer_id, COUNT(DISTINCT city) AS num_cities
FROM superstore_sales
GROUP BY customer_id;

SELECT DISTINCT customer_id, customer_name
FROM superstore_sales
WHERE customer_id IN (
    SELECT customer_id
    FROM customer_unique_city
    WHERE num_cities = 1
);
```

### 2.3.2) Qual o profit, agrupado por categorias, dos top 3 meses de um determinado ano?

```
SELECT month, category, total_profit
FROM profit_by_category_month
WHERE year = 2014 AND month IN (
    SELECT month
    FROM profit_by_category_month
    WHERE year = 2014
    GROUP BY year, month
    ORDER BY SUM(total_profit) DESC
    LIMIT 3
)
ORDER BY month, total_profit DESC;
```

### 2.3.3) Quais são os top 3 produtos que tiveram mais de 20 vendas, mas geraram prejuízo no total?

```

DROP TABLE IF EXISTS product_summary;
CREATE TABLE product_summary (
    product_id TEXT PRIMARY KEY,
    product_name TEXT,
    total_quantity INT,
    total_profit FLOAT);
INSERT INTO product_summary (product_id, product_name, total_quantity, total_profit)
SELECT product_id, product_name, SUM(quantity) AS total_quantity, SUM(profit) AS total_profit
FROM superstore_sales
GROUP BY product_id, product_name;
SELECT * FROM product_summary
WHERE total_quantity > 20 AND total_profit < 0
ORDER BY total_profit ASC
LIMIT 3;

```

2.3.4) Quais os clientes que vivem na região sul e compraram produtos da categoria Office Supplies?

```

SELECT customer_id, customer_name
FROM superstore_sales
WHERE region='South' AND category='Office Supplies';

```

2.3.5) Quais os produtos pedidos pelos 3 clientes que mais compraram (em dólares)?

```

SELECT DISTINCT product_id, product_name
FROM superstore_sales
WHERE customer_id IN ( SELECT customer_id
                        FROM superstore_sales
                        GROUP BY customer_id
                        ORDER BY SUM(sales) DESC
                        LIMIT 3);

```

### Anexo 3 (Neo4j)

#### Anexo 3.1 - Criação e inserção de dados nas Tabelas

```

LOAD CSV WITH HEADERS FROM 'file:///Cliente.csv' AS row
FIELDTERMINATOR ','
WITH row WHERE row.`Customer ID` IS NOT NULL
MERGE (c:Cliente {id: row.`Customer ID`})
SET c.nome = row.`Customer Name`,
    c.segmento = row.Segment

LOAD CSV WITH HEADERS FROM 'file:///ItemPedido.csv' AS row
FIELDTERMINATOR ','
WITH row WHERE row.`Row ID` IS NOT NULL
MERGE (i:ItemPedido {id: row.Row ID})
SET i.pedidoId = row.`Order ID`,
    i.productId = row.`Product ID`,
    i.venda =toFloat(row.Sales),
    i.quantidade = toInteger(row.Quantity),
    i.desconto =toFloat(row.Discount),
    i.lucro =toFloat(row.Profit)

LOAD CSV WITH HEADERS FROM 'file:///Localizacao.csv' AS row
FIELDTERMINATOR ','
WITH row WHERE row.`Location ID` IS NOT NULL
MERGE (l:Localizacao {id: row.`Location ID`})
SET l.cep = row.`Postal Code`,
    l.cidade = row.City,
    l.estado = row.State,
    l.regiao = row.Region,
    l.pais = row.Country

LOAD CSV WITH HEADERS FROM 'file:///Pedido.csv' AS row

```

```

FIELDTERMINATOR ';'
WITH row WHERE row.`Order ID` IS NOT NULL
MERGE (p:Pedido {id: row.`Order ID`})
SET p.dataPedido = row.`Order Date`,
    p.dataEnvio = row.`Ship Date`,
    p.modalidadeEnvio = row.`Ship Mode`,
    p.customerId = row.`Customer ID`,
    p.locationId = row.`Location ID`

LOAD CSV WITH HEADERS FROM 'file:///Produto.csv' AS row
FIELDTERMINATOR ';'
WITH row WHERE row.`Product ID` IS NOT NULL
MERGE (p:Produto {id: row.`Product ID`})
SET p.nome = row.`Product Name`,
    p.categoria = row.Category,
    p.subcategoria = row.`Sub-Category`

MATCH (c:Cliente), (p:Pedido)
WHERE c.id = p.customerId
MERGE (c)-[:FEZ]->(p)

MATCH (i:ItemPedido), (p:Pedido)
WHERE i.pedidoId = p.id
MERGE (p)-[:CONTEM]->(i)

MATCH (i:ItemPedido), (p:Pedido)
WHERE i.pedidoId = p.id
MERGE (i)-[:CONTIDO_EM]->(p)

MATCH (i:ItemPedido), (prod:Produto)
WHERE i.productId = prod.id
MERGE (i)-[:REFERE]->(prod)

MATCH (p:Pedido), (l:Localizacao)
WHERE p.locationId = l.id
MERGE (p)-[:ENVIADO_PARA]->(l)

```

### Anexo 3.2- Queries Neo4j

#### 3.2.1) Quais os produtos pedidos pelos 3 clientes que mais compraram?

```

MATCH (c:Cliente)-[:FEZ]->(p:Pedido)-[:CONTEM]->(i:ItemPedido)
WITH c, sum(i.venda) AS totalComprado
ORDER BY totalComprado DESC
LIMIT 3
WITH collect(c.id) AS topClientes
MATCH (c:Cliente)-[:FEZ]->(:Pedido)-[:CONTEM]->(i:ItemPedido)-[:REFERE]->(prod:Produto)
WHERE c.id IN topClientes
RETURN DISTINCT prod.id AS product_id, prod.nome AS product_name
ORDER BY product_name

```

#### 3.2.2) Quais os clientes que vivem na região Sul e compraram produtos da categoria "Office Supplies"?

```

MATCH (c:Cliente)-[:FEZ]->(p:Pedido)-[:CONTEM]->(i:ItemPedido)-[:REFERE]->(prod:Produto),
      (p)-[:ENVIADO_PARA]->(l:Localizacao)
WHERE l.regiao = 'South' AND prod.categoria = 'Office Supplies'
RETURN DISTINCT c.nome AS Cliente, l.regiao AS Regiao, prod.categoria AS Categotia

```

#### 3.2.3) Quais são os top 3 produtos que tiveram mais de 20 vendas mas geraram prejuízo no total?

```

MATCH (i:ItemPedido)-[:REFERE]->(p:Produto)
WITH p,
    sum(i.quantidade) AS totalQuantidade,
    sum(i.lucro) AS totalLucro
WHERE totalQuantidade > 20 AND totalLucro < 0
RETURN p.nome AS produto, totalQuantidade, totalLucro
ORDER BY totalLucro ASC
LIMIT 3

```

3.2.4) Qual é o profit, agrupado por categorias, dos top 3 meses de um determinado ano?

```

MATCH (i:ItemPedido)-[:CONTIDO_EM]->(p:Pedido)
WHERE p.dataPedido IS NOT NULL
WITH i.lucro AS lucro,
     split(p.dataPedido, "/") AS partesData,
     i, p
WITH
CASE
    WHEN toInteger(partesData[0]) <= 13 THEN toInteger(partesData[0])
    ELSE toInteger(partesData[1])
END AS mes,
CASE
    WHEN toInteger(partesData[2]) >= 1000 THEN toInteger(partesData[2])
    ELSE toInteger(partesData[2]) + 2000
END AS ano,
lucro,
i
WHERE ano = 2013
WITH mes, SUM(lucro) AS lucroTotal
ORDER BY lucroTotal DESC
LIMIT 3
WITH collect(mes) AS topMeses

MATCH (i:ItemPedido)-[:CONTIDO_EM]->(p:Pedido), (i)-[:REFERE]->(prod:Produto)
WHERE p.dataPedido IS NOT NULL
WITH i.lucro AS lucro,
     split(p.dataPedido, "/") AS partesData,
     prod.categoria AS categoria,
     topMeses
WITH
CASE
    WHEN toInteger(partesData[0]) <= 13 THEN toInteger(partesData[0])
    ELSE toInteger(partesData[1])
END AS mes,
CASE
    WHEN toInteger(partesData[2]) >= 1000 THEN toInteger(partesData[2])
    ELSE toInteger(partesData[2]) + 2000
END AS ano,
categoria,
lucro,
topMeses
WHERE ano = 2013 AND mes IN topMeses
RETURN mes, categoria, SUM(lucro) AS lucroTotal
ORDER BY mes, categoria

```

3.2.5) Clientes que pediram sempre para a mesma cidade?

```

MATCH (c:Cliente)-[:FEZ]->(:Pedido)-[:ENVIADO_PARA]->(l:Localizacao)
WITH c, collect(DISTINCT l.cidade) AS cidades
WHERE size(cidades) = 1
RETURN c.id AS clienteId, c.nome AS nomeCliente

```

Anexo 3.3 - Mermaid

O diagrama foi gerado a partir do seguinte código Mermaid.js (graph LR), representando o

modelo de grafos Neo4j com 5 entidades e 5 tipos de relacionamentos. <https://mermaid.live/>

```
graph LR
    %% NÓ CLIENTE
    Cliente["Cliente
    ---
    id
    nome
    segmento"]

    %% NÓ PEDIDO
    Pedido["Pedido
    ---
    id
    dataPedido
    dataEnvio
    modalidadeEnvio
    customerId
    locationId"]

    %% NÓ ITEM_PEDIDO
    ItemPedido["ItemPedido
    ---
    pedidoId
    produtoId
    quantidade
    venda
    desconto
    lucro"]

    %% NÓ PRODUTO
    Produto["Produto
    ---
    id
    nome
    categoria
    subcategoria"]

    %% NÓ LOCALIZAÇÃO
    Localizacao["Localizacao
    ---
    cep
    cidade
    estado
    regiao
    pais"]

    %% RELACIONAMENTOS
    Cliente --"FEZ"--> Pedido
    Pedido --"CONTEM"--> ItemPedido
    ItemPedido --"REFERE"--> Produto
    Pedido --"ENVIADO_PARA"--> Localizacao
    ItemPedido --"CONTIDO_EM"--> Pedido
```