Licenciatura em Engenharia e Gestão de Sistemas de Informação

Universidade do Minho
Escola de Engenharia

Base de Dados 2

2025/2026

Turno PL2 | Grupo 1

"Olist Data Intelligence"

Nome: Ana Íris Machado Torres
Nº: A110489
Email: a110489@uminho.pt

Nome: Daniela Gomes Paulino
Nº: A109519
Email: a109519@uminho.pt

Nome: Maria Leonor Gomes Sampaio
Nº: A111427
Email: a111427@uminho.pt

Nome: Maria Nair da Cunha e Silva Lacerda Ângelo
Nº: A111488
Email: a111488@uminho.pt

Nome: Mariana Silva Gomes
Nº: A110943
Email: a110943@uminho.pt

Índice

1. Introdução 1
2. Dados e Modelo do Domínio da Aplicação 1
2.1. Descrição Geral dos Dados 1
2.2. Modelo Geral dos Dados 3
2.3. Definição de Questões Analíticas Genéricas 4
3. Momentos de Entrega 4
3.1. Momento 1 4
3.1.1. Modelo dos Dados para Implementação 4
3.1.2. Questões Analíticas para Implementação 6
3.1.3 Implementação da Base de Dados 9
3.2. Momento 2 12
3.2.1. Modelo dos Dados para Implementação 12
3.2.2. Questões Analíticas para Implementação 13
3.2.3. Implementação da Base de Dados 15
3.3. Momento 3 19
3.3.1. Modelo dos Dados para Implementação 19
3.3.2. Questões Analíticas para Implementação 20
3.3.3. Implementação da Base de Dados 22
4. Síntese de Resultados 27
5. Avaliação do Grupo de Trabalho 30
6. Referências Bibliográficas 30
ANEXOS 31
ANEXO M1 – SQL 32
ANEXO M2 – Cassandra 38
ANEXO M3 – Neo4j 46

1. Introdução

O presente projeto foi desenvolvido no âmbito da unidade curricular Base de Dados 2, integrada na Licenciatura em Engenharia e Gestão de Sistemas de Informação da Universidade do Minho. Este tem como finalidade a modelação, implementação e exploração analítica de um sistema de informação usando dados reais.

O projeto encontra-se estruturado de forma incremental, contemplando três momentos distintos de avaliação. No primeiro momento **(M1)**, procede-se à modelação conceptual do domínio da aplicação, à sua transformação para modelo relacional e à implementação num Sistema de Gestão de Bases de Dados relacional, assegurando a definição rigorosa de chaves primárias, chaves estrangeiras e restrições de integridade. São igualmente desenvolvidas consultas em SQL destinadas à extração de indicadores analíticos e à validação do modelo implementado.

No segundo momento **(M2)**, o mesmo sistema é reestruturado por uma base de dados do tipo column-family, utilizando a tecnologia Apache Cassandra, permitindo analisar as implicações da modelação orientada a consultas, bem como as diferenças ao nível da normalização, redundância e desempenho.

Por fim, no momento 3 **(M3)**, procede-se à integração e comparação de múltiplas abordagens tecnológicas, envolvendo bases de dados relacionais (SQL), orientadas a colunas (Cassandra) e orientadas a grafos, através do Neo4j. Esta fase permite explorar a representação explícita de relações complexas entre entidades e avaliar a adequação de cada paradigma a diferentes tipos de questões analíticas.

2. Dados e Modelo do Domínio da Aplicação

2.1. Descrição Geral dos Dados

O dataset escolhido para o desenvolvimento do trabalho prático foi o "Brazilian E-Commerce Public Dataset by Olist"[1], disponibilizado na plataforma Kaggle. Trata-se de um conjunto de dados reais referentes a aproximadamente 100 mil encomendas, realizadas entre 2016 e 2018, e efetuadas em múltiplos marketplaces. Este conjunto de dados foi armazenado por uma plataforma central (Olist), que atua como integradora para pequenos e médios retalhistas, estruturado num modelo relacional que possibilita o estudo das vendas através de múltiplas dimensões. Isto permite analisar não só o volume de vendas, mas também a performance de diferentes vendedores e a eficiência logística em diversas regiões do Brasil.

- **Customer –** A entidade Customer agrupa as informações relativas aos clientes que realizam compras através da plataforma.

Atributo	Significado	Valores Possíveis	Exemplo de Valores Disponíveis nos Dados
customer_id	Chave gerada para cada encomenda. É o identificador que liga o cliente à tabela Order.	Cadeia alfanumérica (32 caracteres)	06b8999e2fba1a1fbc88172c00ba8bc7
customer_unique_id	Identificador real e único do cliente.	Cadeia alfanumérica (32 caracteres)	861eff4711a542e4b93843c6dd7febb0
customer_zip_code_prefix	Os primeiros 5 dígitos do código postal do cliente.	Numérico/Texto (5 caracteres)	14409
customer_city	Nome da cidade onde o cliente reside ou mandou entregar.	Cadeia de caracteres (Texto)	franca
customer_state	Sigla do estado onde o cliente reside.	Sigla de 2 caracteres	SP

Tabela 1 – Dicionário de dados da entidade Customer

[1] MAGIOLO,F., DABAGUE,L., SIONEK,A.(2022), *Brazilian E-Commerce Public Dataset by Olist, Kaggle.* Consultado em https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce?resource=download&select=olist_sellers_dataset.csv a 19 de fevereiro de 2026.

- **Product –** A entidade Product constitui o catálogo de bens disponíveis no marketplace. Para além da categorização dos produtos, esta entidade armazena características físicas críticas, como peso e dimensões volumétricas.

Atributo	Significado	Valores Possíveis	Exemplo de Valores Disponíveis nos Dados
product_id	Identificador único de cada produto.	Cadeia alfanumérica (32 caracteres)	1e9e8ef04dbcff4541ed26657ea517e5
product_category_name	Nome da categoria principal do produto.	Cadeia de caracteres (Lista de categorias)	perfumaria
product_name_lenght	Quantidade de caracteres contidos no nome do produto.	Inteiro (entre 5 e 76)	40
product_description_lenght	Quantidade de caracteres contidos na descrição do produto.	Inteiro (entre 4 e 3992)	287
product_photos_qty	Número de fotografias do produto publicadas no anúncio.	Inteiro (entre 1 e 20)	1
product_weight_g	Peso do produto medido em gramas.	Inteiro (entre 0 e 40.4g)	225
product_length_cm	Comprimento do produto em centímetros.	Inteiro (entre 7 e 105)	16
product_height_cm	Altura do produto em centímetros.	Inteiro (entre 2 e 105)	10
product_width_cm	Largura do produto em centímetros.	Inteiro (entre 6 e 118)	14

Tabela 2 – Dicionário de dados da entidade Product

- **Order –** A entidade Order representa a venda, funciona como o núcleo central do modelo de dados, registando cada transação efetuada.

Atributo	Significado	Valores Possíveis	Exemplo de Valores Disponíveis nos Dados
order_id	Identificador único da encomenda (venda).	Cadeia alfanumérica (32 caracteres)	e481f51cbdc54678b7cc49136f2d6af7
customer_id	Chave gerada para cada encomenda. É o identificador que liga o cliente à tabela Order.	Cadeia alfanumérica (32 caracteres)	9ef432eb6251297304e76186b10a928d
order_status	Estado atual da encomenda no ciclo de venda.	delivered; shipping; invoiced; unavailable; canceled; created	delivered
order_purchase_timestamp	Data e hora exata em que o cliente efetuou a compra.	Data/Hora (AAAA-MM-DD HH:MM:SS)	2017-10-02 10:56:33
order_approved_at	Data e hora em que o pagamento foi aprovado pelo sistema.	Data/Hora (AAAA-MM-DD HH:MM:SS)	2017-10-02 11:07:15
order_delivered_carrier_date	Data e hora em que a encomenda foi entregue à transportadora.	Data/Hora (AAAA-MM-DD HH:MM:SS)	2017-10-04 19:55:00
order_delivered_customer_date	Data e hora em que o cliente recebeu efetivamente o produto.	Data/Hora (AAAA-MM-DD HH:MM:SS)	2017-10-10 21:25:13
order_estimated_delivery_date	Data de entrega prevista comunicada ao cliente no momento da compra.	Data/Hora (AAAA-MM-DD HH:MM:SS)	2017-10-18 00:00:00

Tabela 3 – Dicionário de dados da entidade Order

- **Order_item –** A entidade Order_item detalha o conteúdo de cada encomenda, uma vez que uma única venda pode conter múltiplos produtos.

Atributo	Significado	Valores Possíveis	Exemplo de Valores Disponíveis nos Dados
order_item_id	Número sequencial que identifica a ordem dos produtos dentro de uma mesma encomenda.	Inteiro (1 a 21)	1
order_id	Identificador único da encomenda (venda).	Cadeia alfanumérica (32 caracteres)	00010242fe8c5a6d1ba2dd792cb16214
product_id	Identificador único do produto	Cadeia alfanumérica (32 caracteres)	4244733e06e7ecb4970a6e2683c13e61
seller_id	Identificador único do vendedor	Cadeia alfanumérica (32 caracteres)	48436dade18ac8b2bce089ec2a041202
shipping_limit_date	Data e hora limite para o vendedor entregar o produto à transportadora	Data/Hora (AAAA-MM-DD HH:MM:SS)	2017-09-19 09:45:35
price	Preço unitário do produto.	Decimal Positivo (0.85 a 6.74 mil)	58.90
freight_value	Valor da taxa de envio para este produto específico. com base nas suas medidas e peso. A taxa total da encomenda obtém-se somando os valores de todos os produtos.	Decimal Positivo (0 a 410)	13.29

Tabela 4 – Dicionário de dados da entidade Order_item

- **Seller –** A entidade Seller representa os retalhistas parceiros que utilizam a infraestrutura da Olist para comercializar os seus produtos.

Atributo	Significado	Valores Possíveis	Exemplo de Valores Disponíveis nos Dados
seller_id	Identificador único de cada vendedor.	Cadeia alfanumérica (32 caracteres)	3442f8959a84dea7ee197c632cb2df15
seller_zip_code_prefix	Primeiros cinco dígitos do código postal da localização do vendedor.	Numérico/Texto (5 caracteres)	13023
seller_city	Nome da cidade onde o vendedor está sediado.	Cadeia de caracteres (Texto)	campinas
seller_state	Sigla do estado onde o vendedor está sediado.	Sigla de 2 caracteres	SP

Tabela 5 – Dicionário de dados da entidade Seller

2.2. Modelo Geral dos Dados

Assunções de Domínio:

- Um Customer só é considerado como tal se tiver efetuado pelo menos um pedido (Order).
- Um Seller só é registado no sistema se possuir pelo menos um produto associado a um item de venda (Order_item).
- As datas nulas (vazias) em campos como order_delivered_customer_date indicam que a encomenda ainda não atingiu essa fase do ciclo de vida ou foi cancelada.

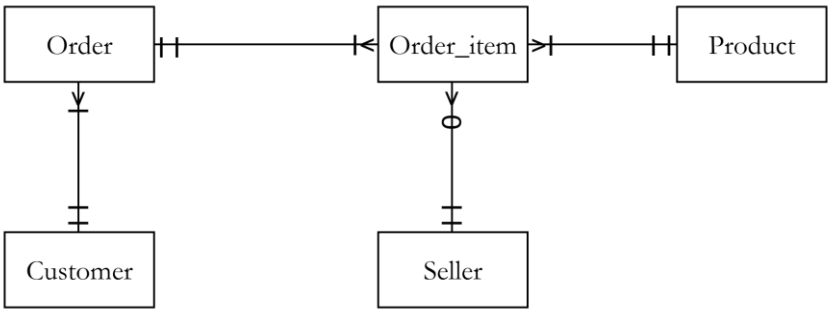

Figura 1 – Modelo de Dados do Domínio de Aplicação

2.3. Definição de Questões Analíticas Genéricas

Questão	Dados e horizonte temporal a considerar
1. Quais são as 10 categorias de produtos que geram maior receita total (preço + taxa de envio) para a plataforma?	Ranking de categorias da tabela Product pelo somatório de valores da Order_item entre 2017 e 2018.
2. Qual é o produto mais vendido ("Best Seller") em termos de quantidade em cada estado do Brasil?	Contagem de product_id agrupado por estado da tabela Customer em todo o período disponível.
3. Qual é o fluxo de vendas entre diferentes estados (Vendedor vs. Cliente) para identificar as rotas comerciais mais frequentes?	Contagem de transações a fim de interligar o estado da tabela Seller com o estado da tabela Customer no ano de 2016
4. Qual é a taxa de sucesso das vendas, comparando encomendas entregues e canceladas, por categoria de produto e faturação perdida?	Rácio de estados 'delivered' e 'canceled' da tabela Order e somatório de preço e taxa de envio das vendas canceladas por categoria (Product) em 2017 e 2018.
5. Quais são os 10 clientes que geram maior receita total (preço + taxa de envio) para plataforma?	Ranking de customer_unique_id pelo somatório de preço e taxa de envio das tabelas Order e Order_item entre 2016 e 2018.
6. Quais os produtos que são frequentemente comprados em conjunto por categoria na mesma encomenda?	Identificação de pares de product_id e as suas categorias (Product) que partilham o mesmo order_id na Order_item entre 2016 e 2018.

Tabela 6 – Questões Analíticas

3. Momentos de Entrega

3.1. Momento 1

3.1.1. Modelo dos Dados para Implementação

O modelo de dados para implementação foi desenvolvido a partir do modelo conceptual do domínio da aplicação apresentado na Secção 2.2. No âmbito do Momento 1 (M1), este modelo conceptual foi transformado num **modelo relacional**, adequado à implementação num Sistema de Gestão de Bases de Dados relacional.

Durante este processo, cada entidade do domínio foi convertida numa tabela da base de dados, sendo cuidadosamente definidas as respetivas **chaves primárias** e **chaves estrangeiras**, de modo a garantir a integridade de entidade, a integridade referencial e a consistência global dos dados armazenados.

Order_item (order_id, order_item_id, product_id, seller_id, shipping_limit_date, price, freight_value)

Customer (customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state)

Seller (seller_id, seller_zip_code_prefix, seller_city, seller_state)

Product (product_id, product_category_name, product_name_length, product_description_length, product_photos_qty, product_weight_g, product_length_cm, product_height_cm, product_width_cm)

Order (order_id, customer_id, order_status, order_purchase_timestamp, order_approved_at, order_delivered_carrier_date, order_delivered_customer_date, order_estimated_delivery_date)

A Figura 2 apresenta o modelo de relacional de dados que será implementado na base de dados:

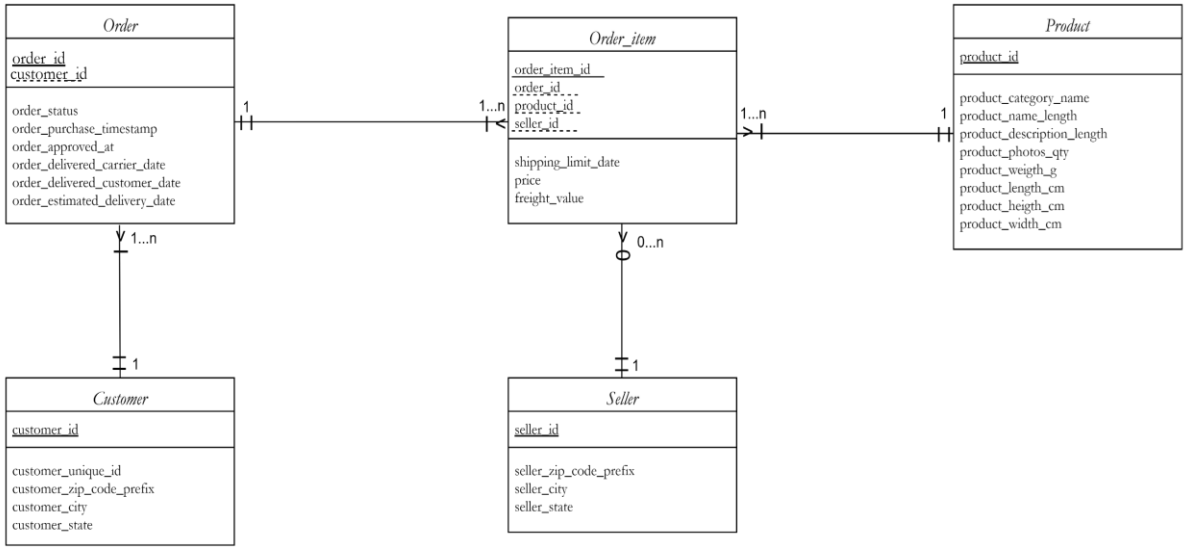

Figura 2 – Modelo de Dados para M1

O modelo de dados é constituído pelas tabelas Customer, Seller, Product, Order e Order_item, refletindo diretamente as entidades e relacionamentos definidos no domínio da aplicação.

A tabela Customer armazena a informação relativa aos clientes da plataforma, enquanto a tabela Seller contém os dados dos vendedores. A tabela Product representa o catálogo de produtos disponíveis no marketplace.

A tabela Order regista as encomendas efetuadas pelos clientes, encontrando-se associada à tabela Customer através da chave estrangeira customer_id. Por sua vez, a tabela Order_item detalha os produtos incluídos em cada encomenda, estabelecendo relações com as tabelas Order, Product e Seller através das respetivas chaves estrangeiras.

A relação entre encomendas e produtos é implementada através da tabela Order_item, que funciona como uma tabela associativa, permitindo representar uma relação do tipo muitos-para-muitos entre encomendas e produtos. Esta abordagem assegura flexibilidade na representação dos dados e suporta a execução das consultas analíticas definidas para o projeto.

O modelo relacional obtido servirá, assim, de base para a criação da base de dados no M1 e para a implementação das consultas analíticas, garantindo coerência com os objetivos definidos e com as questões analíticas genéricas identificadas na Secção 2.3.

- **Chaves Primárias e Estrangeiras**

Tabela	Chave Primária	Chaves Estrangeiras
Customer	Customer_id	_
Seller	seller_id	_
Product	Product_id	_
Order	Order_id	Customer_id-customer(customer_id)
Order_item	Order_id, order_item_id	Order_id-order(order_id) Product_id-product(product_id) Seller_id-seller(seller_id)

Tabela 7 – Definição de Chaves Primárias e Chaves Estrangeiras para o modelo relacional.

- **Restrições de Integridade**

Tipo de Restrição	Descrição
Integridade de Entidade	Todas as tabelas possuem chave primária definida
Integridade Referencial	Encomendas e itens de encomenda referenciam apenas clientes, produtos e vendedores válidos
Integridade de Campos	Identificadores e atributos essenciais definidos como obrigatórios

Tabela 8 – Matriz de restrições de integridade

- **Restrições de Domínio**

Atributo	Regra Considerada
Order_status	Representa o estado da encomenda no ciclo de vida
Price	Valor monetário associado ao produto
Freight_value	Valor da taxa de envio para o produto
Datas de entrega	Podem assumir valores nulos para encomendas não entregues ou canceladas

Tabela 9 – Regras e restrições de domínio aplicadas aos atributos

- **Implementação de Base de Dados**

Aspeto	Descrição
Ferramenta utilizada	DBeaver
Linguagem	SQL
Tipo de Comando	Criação de tabelas e relações
Código Completo	Incluído na 3.2

Tabela 10 – Resumo dos aspetos técnicos da implementação em SQL

A implementação da base de dados foi realizada num sistema de Gestão de bases de dados relacional, recorrendo à linguagem SQL para a criação das tabelas, definição das chaves primárias e estrangeiras, bem como das respetivas restrições de integridade.

O código SQL completo utilizado na criação do esquema da base dados encontra-se disponível no ponto 3.2 do projeto.

- **Adequação às Questões Analíticas**

Dimensão Analítica	Tabelas Envolvidas
Análise por Cliente	Customer, order, order_item
Análise por Vendedor	Seller, order_item
Análise por Produto/categoria	Product, order_item
Análise Temporal	Order
Análise Geográfica	Customer, seller

Tabela 11 – Mapeamento das dimensões analíticas e tabelas envolvidas

3.1.2. Questões Analíticas para Implementação

Questão 1 - Quais são as 10 categorias de produtos que geram maior receita total (preço + taxa de envio) para a plataforma?

```sql
CREATE VIEW Receita_por_categoria AS
SELECT p.product_category_name,
          SUM(oi.price + oi.freight_value) AS receita_total
FROM Order_item oi
JOIN Product p ON oi.product_id = p.product_id
GROUP BY p.product_category_name;

SELECT *
FROM Receita_por_categoria
ORDER BY receita_total DESC
LIMIT 10;
```

product(+) 1 | Estatísticas 1

CREATE VIEW Receita_por_categoria AS SELE | Insira uma expressão

	product_category_name	receita_total
1	beleza_saude	1.441.248,07
2	relogios_presentes	1.305.541,61
3	cama_mesa_banho	1.241.681,72
4	esporte_lazer	1.156.656,48
5	informatica_acessorios	1.059.272,4
6	moveis_decoracao	902.511,79
7	utilidades_domesticas	778.397,77
8	cool_stuff	719.329,95
9	automotivo	685.384,32
10	ferramentas_jardim	584.219,21

Atualizar | Salvar | Cancelar

Tabela 12 – Top 10 categorias de produtos com maior receita total acumulada

Questão 2 - Qual é o produto mais vendido ("Best Seller") em termos de quantidade em cada estado do Brasil?

```sql
SELECT
    c.customer_state AS Estado,
    p.product_id AS ID_Produto,
    p.product_category_name AS Categoria,
    COUNT(oi.product_id) AS Quantidade_Vendida
FROM Customer c
JOIN `Order` o ON c.customer_id = o.customer_id
JOIN Order_item oi ON o.order_id = oi.order_id
JOIN Product p ON oi.product_id = p.product_id
GROUP BY c.customer_state, p.product_id, p.product_category_name
HAVING (c.customer_state, COUNT(oi.product_id)) IN (
    -- esta subquery encontra a quantidade máxima vendida em cada estado
    SELECT estado_sub, MAX(total_vendas)
    FROM (
        SELECT
            c2.customer_state AS estado_sub,
            COUNT(oi2.product_id) AS total_vendas
        FROM Customer c2
        JOIN `Order` o2 ON c2.customer_id = o2.customer_id
        JOIN Order_item oi2 ON o2.order_id = oi2.order_id
        GROUP BY c2.customer_state, oi2.product_id
    ) AS sub_tabela
    GROUP BY estado_sub
```

```
)
ORDER BY Quantidade_Vendida DESC;
```

	Estado	ID_Produto	Categoria	Quantidade_Vendida
1	SP	aca2eb7d00ea1a7b8ebd4e68314663af	moveis_decoracao	265
2	RJ	d1c427060a0f73f6b889a5c7c61f2ac4	informatica_acessorios	102
3	MG	d1c427060a0f73f6b889a5c7c61f2ac4	informatica_acessorios	82
4	RS	422879e10f46682990de24d770e7f83d	ferramentas_jardim	42
5	PR	a62e25e09e05e6faf31d90c6ec1aa3d1	relogios_presentes	28
6	SC	368c6c730842d78016ad823897a372db	ferramentas_jardim	27

Tabela 13 – Identificação do produto Best Seller por estado

Questão 3. Qual é o fluxo de vendas entre diferentes estados (Vendedor vs. Cliente) para identificar as rotas comerciais mais frequentes?

```
SELECT c.customer_state, s.seller_state,  COUNT(distinct
o.order_id) AS total_fluxo
FROM Customer c
JOIN `Order` o ON c.customer_id=o.customer_id
JOIN Order_item oi ON o.order_id=oi.order_id
JOIN Seller s ON oi.seller_id=s.seller_id
WHERE o.order_purchase_timestamp BETWEEN '2016-01-01
00:00:00' AND '2016-12-31 23:59:59'
GROUP BY s.seller_state, c.customer_state
ORDER BY total_fluxo DESC;
```

	customer_state	seller_state	total_fluxo
1	SP	SP	69
2	MG	SP	22
3	RJ	SP	21
4	RJ	RJ	15
5	SP	PR	13
6	RS	SP	12
7	SP	RJ	12
8	PR	SP	9
9	PR	PR	7
10	SC	SP	6
11	MG	MG	6
12	MG	RJ	5
13	SP	SC	5
14	RJ	PR	5
15	SP	MG	5
16	CE	SP	4
17	PE	SP	4
18	RJ	MG	4
19	BA	SP	3
20	DF	SP	3
21	RS	SC	3
22	RS	RS	3
23	GO	SP	3
24	MA	SP	3
25	PR	MG	3
26	PA	SP	3

Tabela 14 – Fluxo comercial entre estados

Questão 4 - Qual é a taxa de sucesso das vendas, comparando encomendas entregues e canceladas, por categoria de produto e faturação perdida?

```
SELECT
    p.product_category_name AS Categoria,
    SUM(IF(o.order_status = 'delivered', 1, 0)) AS Total_Entregue,
    SUM(IF(o.order_status = 'canceled', 1, 0)) AS Total_Cancelado,
    -- Cálculo da Taxa de Sucesso: (Entregues / (Entregues + Cancelados))
    ROUND(
        SUM(IF(o.order_status = 'delivered', 1, 0)) /
            NULLIF(SUM(IF(o.order_status = 'delivered' OR o.order_status = 'canceled', 1, 0)), 0) * 100,
2) AS Taxa_Sucesso_Percent,
    -- Soma do preço + taxa de envio apenas para os cancelados (Faturação Perdida)
    SUM(IF(o.order_status = 'canceled', oi.price + oi.freight_value, 0)) AS Faturacao_Perdida
FROM
    Order_item oi
JOIN
    `Order` o ON oi.order_id = o.order_id
JOIN
    Product p ON oi.product_id = p.product_id
```

```sql
WHERE
    YEAR(o.order_purchase_timestamp) IN (2017, 2018)
GROUP BY
    p.product_category_name
HAVING
    Total_Entregue > 0 OR Total_Cancelado > 0
ORDER BY
    Faturacao_Perdida DESC;
```

	Categoria	Total_Entregue	Total_Cancelado	Taxa_Sucesso_Percent	Faturacao_Perdida
1	cool_stuff	3.711	16	99,57	15.153,48
2	informatica_acessorios	7.632	46	99,4	8.330,82
3	esporte_lazer	8.414	49	99,42	7.975,5
4	relogios_presentes	5.855	21	99,64	7.249,14
5	utilidades_domesticas	6.783	49	99,28	6.362,63
6	automotivo	4.132	28	99,33	6.272,58
7	ferramentas_jardim	4.263	19	99,56	4.694,01
8	brinquedos	4.006	31	99,23	4.194,71
9	instrumentos_musicais	651	11	98,34	4.116,05
10	beleza_saude	9.422	34	99,64	3.397,85
11	eletroportateis	658	8	98,8	2.937,33
12	moveis_decoracao	8.095	36	99,56	2.885,75
13	eletrodomesticos_2	231	3	98,72	2.862,4
14	consoles_games	1.082	10	99,08	2.653,64
15	eletronicos	2.728	11	99,6	2.180,51
16	perfumaria	3.311	13	99,61	1.849,6
17	telefonia_fixa	250	3	98,81	1.806,76
18	papelaria	2.466	12	99,52	1.589,53
19	bebes	2.971	19	99,36	1.496,47
20	dvds_blu_ray	61	2	96,83	1.483,76
21	livros_interesse_geral	537	7	98,71	1.393,71
22	cama_mesa_banho	10.945	18	99,84	1.295,59
23	telefonia	4.422	16	99,64	1.250,05
24	fashion_bolsas_e_aces	1.976	5	99,75	1.185,27
25		1.535	14	99,1	1.180,8

Tabela 15 – Análise da taxa de sucesso de vendas e faturação perdida por categoria

Questão 5 - Quais são os 10 clientes que geram maior receita total (preço + taxa de envio) para plataforma?

```sql
SELECT c.customer_unique_id,
    SUM(oi.price + oi.freight_value) AS receita_total
FROM `Order` o
JOIN customer c
    ON c.customer_id = o.customer_id
JOIN order_item oi
    ON oi.order_id = o.order_id
WHERE o.order_purchase_timestamp
      BETWEEN '2016-01-01 00:00:00'
          AND '2018-12-31 23:59:59'
  AND o.order_status = 'delivered'
GROUP BY c.customer_unique_id
ORDER BY receita_total DESC
LIMIT 10;
```

	customer_unique_id	receita_total
1	0a0a92112bd4c708ca5fde585afaa872	13,664.08
2	da122df9eeddfedc1dc1f5349a1a690c	7,571.63
3	763c8b1c9c68a0229c42c9fc6f662b93	7,274.88
4	dc4802a71eae9be1dd28f5d788ceb526	6,929.31
5	459bef486812aa25204be022145caa62	6,922.21
6	ff4159b92c40ebe40454e3e6a7c35ed6	6,726.66
7	4007669dec559734d6f53e029e360987	6,081.54
8	eebb5dda148d3893cdaf5b5ca3040ccb	4,764.34
9	48e1ac109decbb87765a3eade6854098	4,681.78
10	c8460e4251689ba205045f3ea17884a1	4,655.88

Tabela 16 – Ranking dos 10 clientes com maior volume financeiro gerado

Questão 6 - Quais os produtos que são frequentemente comprados em conjunto por categoria na mesma encomenda?

```sql
CREATE OR REPLACE VIEW vw_pares_produtos_exatos AS
SELECT
    oi1.product_id AS id_produto_A,
    oi2.product_id AS id_produto_B,
    COUNT(*) AS vezes_comprados_juntos
FROM
    Order_item oi1
JOIN
    Order_item oi2 ON oi1.order_id = oi2.order_id
WHERE
    oi1.product_id < oi2.product_id
```

```sql
GROUP BY
    oi1.product_id, oi2.product_id
ORDER BY
    vezes_comprados_juntos DESC;
SELECT
    oi1.product_id,
    p1.product_category_name,
    oi2.product_id,
    p2.product_category_name,
    COUNT(*) AS frequencia
FROM Order_item oi1
JOIN Order_item oi2 ON oi1.order_id = oi2.order_id
JOIN Product p1 ON oi1.product_id = p1.product_id
JOIN Product p2 ON oi2.product_id = p2.product_id
WHERE oi1.product_id < oi2.product_id
GROUP BY oi1.product_id, oi2.product_id, p1.product_category_name, p2.product_category_name
ORDER BY frequencia DESC
LIMIT 10;
```

order_item(+) 1 | Estatísticas 1

SELECT oi1.product_id, p1.product_category_

Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	product_id	product_category_name	product_id	product_category_name	frequencia
1	05b515fdc76e888aada3c6d66c201dff	beleza_saude	270516a3f41dc035aa87d220228f844c	beleza_saude	100
2	36f60d45225e60c7da4558b070ce4b60	informatica_acessorios	e53e557d5a159f5aa2c5e995dfdf244b	informatica_acessorios	48
3	710b7c26b7a742f497bba45fab91a25f	utilidades_domesticas	a9d9db064d4afd4458eb3e139fe29167	utilidades_domesticas	36
4	62995b7e571f5760017991632bbfd311	utilidades_domesticas	ac1ad58efc1ebf66bfadc09f29bdedc0	utilidades_domesticas	36
5	35afc973633aaeb6b877ff57b2793310	casa_conforto	99a4788cb24856965c36a24e339b6058	cama_mesa_banho	30
6	308e4e21ae228a10f6370a243ae59995	moveis_escritorio	90b58782fdd04cb829667fcc41fb65f5	moveis_escritorio	30
7	2ef36e1cae01b86d0ff0a2f50ff2bd53	moveis_decoracao	53759a2ecddad2bb87a079a1f1519f73	ferramentas_jardim	30
8	18486698933fbb64af6c0a255f7dd64c	informatica_acessorios	dbb67791e405873b259e4656bf971246	informatica_acessorios	26
9	58efb9b638561ce132216a9a612513e2	moveis_sala	872db866d615db59612ac933f43d6b22	moveis_sala	25
10	0db4869ac8a908414823d44d742f5ba3	moveis_escritorio	47920da896bbd89e851429c78a7a7a8b	moveis_escritorio	24

Tabela 17 – Listagem dos pares de produtos mais vendidos em conjunto, detalhados por identificador único (ID) e nome da categoria.

3.1.3 Implementação da Base de Dados

A implementação da base de dados constituiu a etapa central do projeto, o que permitiu transformar um conjunto extenso de ficheiros CSV num sistema relacional coerente, íntegro e preparado para responder às questões analíticas definidas. Todo o processo foi desenvolvido no MySQL, executado num ambiente Docker, e gerido através do DBeaver, o que facilitou tanto a criação das estruturas como a validação dos dados.

A primeira fase consistiu na definição da arquitetura das tabelas, o que garantiu que o modelo relacional refletia corretamente a lógica do domínio. As entidades base, Customer, Seller e Product, foram criadas inicialmente, uma vez que não dependem de outras tabelas e servem de suporte às restantes relações. Cada uma foi estruturada com uma chave primária do tipo VARCHAR(32), o que assegurou a identificação única dos registos. Posteriormente, foi criada a tabela Order, associada aos clientes através de uma chave estrangeira.

Nessa tabela, foi necessário utilizar backticks (plicas) para delimitar o nome da tabela Order, uma vez que este termo coincide com uma palavra reservada do MySQL. Esta prática evitou conflitos com o interpretador SQL e garantiu que o identificador é tratado corretamente como o nome da tabela.

E por fim a tabela Order_item, que integra simultaneamente dependências de Order, Product e Seller. Esta última inclui uma chave primária composta (order_id, order_item_id), necessária para distinguir cada item individual dentro de uma mesma encomenda. Um excerto representativo desta fase pode ser observado no bloco de criação da tabela Order_item:

Sql:

```sql
CREATE TABLE Order_item (
    order_id VARCHAR(32) NOT NULL,
    order_item_id INT NOT NULL,
    product_id VARCHAR(32) NOT NULL,
```

```sql
    seller_id VARCHAR(32) NOT NULL,
    shipping_limit_date DATETIME,
    price DECIMAL(10,2),
    freight_value DECIMAL(10,2),
    PRIMARY KEY (order_id, order_item_id),
    FOREIGN KEY (order_id) REFERENCES `Order`(order_id),
    FOREIGN KEY (product_id) REFERENCES Product(product_id),
    FOREIGN KEY (seller_id) REFERENCES Seller(seller_id)
);
```

Concluída a estrutura relacional, avançou-se para a preparação e importação dos dados. Os ficheiros CSV foram colocados numa diretoria partilhada com o contentor Docker, sendo necessário ajustar permissões no sistema operativo para garantir que o MySQL conseguia aceder aos ficheiros sem restrições. Esta etapa revelou-se particularmente exigente devido à presença de valores em branco em colunas numéricas e de datas, o que originava erros como Error 1366 e Error 1292 durante a execução do script.

Para ultrapassar estes problemas, foi adotada uma abordagem de pré-processamento em tempo real utilizando o comando LOAD DATA INFILE com variáveis temporárias. Cada valor era inicialmente lido para uma variável (por exemplo, @v_purchase ou @vweight) e apenas depois convertido para o tipo adequado, substituindo automaticamente strings vazias por NULL. Esta solução permitiu contornar as inconsistências do dataset original sem necessidade de alterar os ficheiros CSV.

Importa referir que o desenvolvimento desta lógica de tratamento de dados contou com o apoio de Inteligência Artificial, que auxiliou na criação de condições e expressões IF() capazes de validar e corrigir automaticamente valores inválidos durante a importação. Este recurso revelou-se particularmente útil para acelerar o processo e garantir que a importação dos dados fosse robusta e consistente. Um exemplo desta técnica pode ser observado na tabela Order:

Sql:

```sql
SET
order_purchase_timestamp = IF(@v_purchase = '', NULL, @v_purchase),
order_approved_at = IF(@v_approved = '', NULL, @v_approved),
order_delivered_carrier_date = IF(@v_carrier = '', NULL, @v_carrier),
order_delivered_customer_date = IF(@v_customer = '', NULL, @v_customer),
order_estimated_delivery_date = IF(@v_estimated = '', NULL, @v_estimated);
```

Graças a este mecanismo, foi possível carregar com sucesso os registos, o que demonstrou a eficiência da solução implementada e a sua robustez.

Com a base de dados devidamente estruturada e carregada, a fase final da implementação consistiu no desenvolvimento técnico das seis consultas SQL destinadas a responder aos objetivos analíticos definidos. Esta etapa exigiu a aplicação de diferentes técnicas, organizadas segundo uma lógica de complexidade crescente, o que garantiu que cada critério fosse calculado com rigor e alinhado com as necessidades definidas.

A resolução das questões iniciou-se com o conjunto de análises orientadas para o desempenho financeiro, correspondentes às **Questões 1, 4 e 5**.

Estas consultas centraram-se na transformação de dados transacionais em indicadores de receita, o que permitiu identificar padrões de rentabilidade e comportamento de compra.

Na **Questão 1**, foi criada uma View (Receita_por_categoria) que junta o valor total de cada venda, através da soma entre o preço do produto e o valor de transporte (price + freight_value). Esta abordagem, permitiu criar de forma imediata um ranking das categorias mais lucrativas. Já na **Questão 4**, recorreu-se a uma lógica condicional com SUM(IF()) para ser possível distinguir entre encomendas entregues e canceladas, o que possibilitou o cálculo simultâneo da taxa de sucesso e da faturação perdida por categoria. O seguinte excerto ilustra essa técnica:

Sql:

```sql
SUM(IF(o.order_status = 'canceled', oi.price + oi.freight_value, 0)) AS Faturacao_Perdida
```

Na **Questão 5**, a análise incidiu sobre a receita concebida por cliente, que utilizou o campo customer_unique_id para evitar duplicações e filtrar exclusivamente transações com estado delivered, o que assegurou que apenas vendas efetivamente concluídas contribuíam para o cálculo final.

As Questões 2 e 3 abordaram a dimensão geográfica e os fluxos de mercado, o que exigiu a junção de múltiplas tabelas para relacionar a localização física de clientes e vendedores com os padrões de venda.

Na **Questão 2**, dedicada à identificação do best seller por estado brasileiro, foi utilizada uma subquery que isolava previamente o valor máximo de vendas por estado, o que garantiu que a consulta principal apresentava apenas o produto líder em cada região. O mecanismo pode ser observado no seguinte excerto:

Sql:

```
HAVING (c.customer_state, COUNT(oi.product_id)) IN (
    SELECT estado_sub, MAX(total_vendas)
    FROM (...)
    GROUP BY estado_sub
)
```

Já na **Questão 3**, estabeleceu-se uma ligação direta entre a tabela Seller (origem) e Customer (destino) através do order_id, o que permitiu quantificar as rotas comerciais mais frequentes no ano de 2016 e compreender a dinâmica logística entre regiões.

A implementação da **Questão 6** teve como objetivo identificar produtos frequentemente comprados em conjunto, analisando a concorrência de itens dentro da mesma encomenda. Para isso, foi utilizado um self-join na tabela Order_item, relacionando-a consigo própria através do order_id, o que permitiu detetar todos os pares de produtos presentes no mesmo carrinho. Para evitar duplicações, como contar A-B e B-A como pares distintos, foi aplicada a condição oi1.product_id < oi2.product_id, o que garantiu que cada combinação fosse registada apenas uma vez.

Para otimizar o processo e facilitar consultas posteriores, esta lógica foi encapsulada numa View (vw_pares_produtos_exatos), que armazena os pares de produtos e o número de vezes em que surgem juntos:

Sql:

```
CREATE OR REPLACE VIEW vw_pares_produtos_exatos AS
SELECT
    oi1.product_id AS id_produto_A,
    oi2.product_id AS id_produto_B,
    COUNT(*) AS vezes_comprados_juntos
FROM Order_item oi1
JOIN Order_item oi2 ON oi1.order_id = oi2.order_id
WHERE oi1.product_id < oi2.product_id
GROUP BY oi1.product_id, oi2.product_id;
```

A análise foi depois enriquecida com a junção à tabela Product, o que permitiu identificar as categorias envolvidas e revelar padrões de compra entre produtos de diferentes segmentos, que é informação útil para estratégias de cross-selling. O exemplo seguinte mostra a extração dos pares mais frequentes:

Sql:

```
SELECT
    oi1.product_id, p1.product_category_name,
    oi2.product_id, p2.product_category_name,
    COUNT(*) AS frequencia
FROM Order_item oi1
JOIN Order_item oi2 ON oi1.order_id = oi2.order_id
JOIN Product p1 ON oi1.product_id = p1.product_id
JOIN Product p2 ON oi2.product_id = p2.product_id
WHERE oi1.product_id < oi2.product_id
GROUP BY oi1.product_id, oi2.product_id
ORDER BY frequencia DESC
LIMIT
```

A combinação de diferentes técnicas, permitiu construir um conjunto de consultas robustas, capazes de responder de forma precisa e eficiente às seis questões analíticas propostas.

Todo o código integral utilizado nesta fase encontra-se incluído no **Anexo M1 – SQL**.

3.2. Momento 2

3.2.1. Modelo dos Dados para Implementação

Para a implementação em Apache Cassandra (M2), a transformação do modelo relacional exigiu uma mudança de paradigma para a abordagem "Query design (first!)". Em vez de modelar entidades e utilizar junções complexas para a leitura, os dados originais foram denormalizados e as tabelas (Column Families) foram desenhadas especificamente para fornecer a resposta exata a cada questão analítica de forma direta e otimizada.

A otimização destas Column Families assenta na definição estratégica da sua Chave Primária, dividida em duas componentes:

- **Partition Key (Row Key):** Define como os dados são distribuídos e agrupados nas partições do cluster.
- **Clustering Column:** Garante a ordenação física e contínua dos dados no disco logo no momento da escrita, eliminando a necessidade de usar funções de ordenação dispendiosas durante a leitura.

A Figura 3 ilustra o modelo de dados desenhado para este momento, representando visualmente as estruturas criadas para responder às questões propostas na secção 3.2.3.

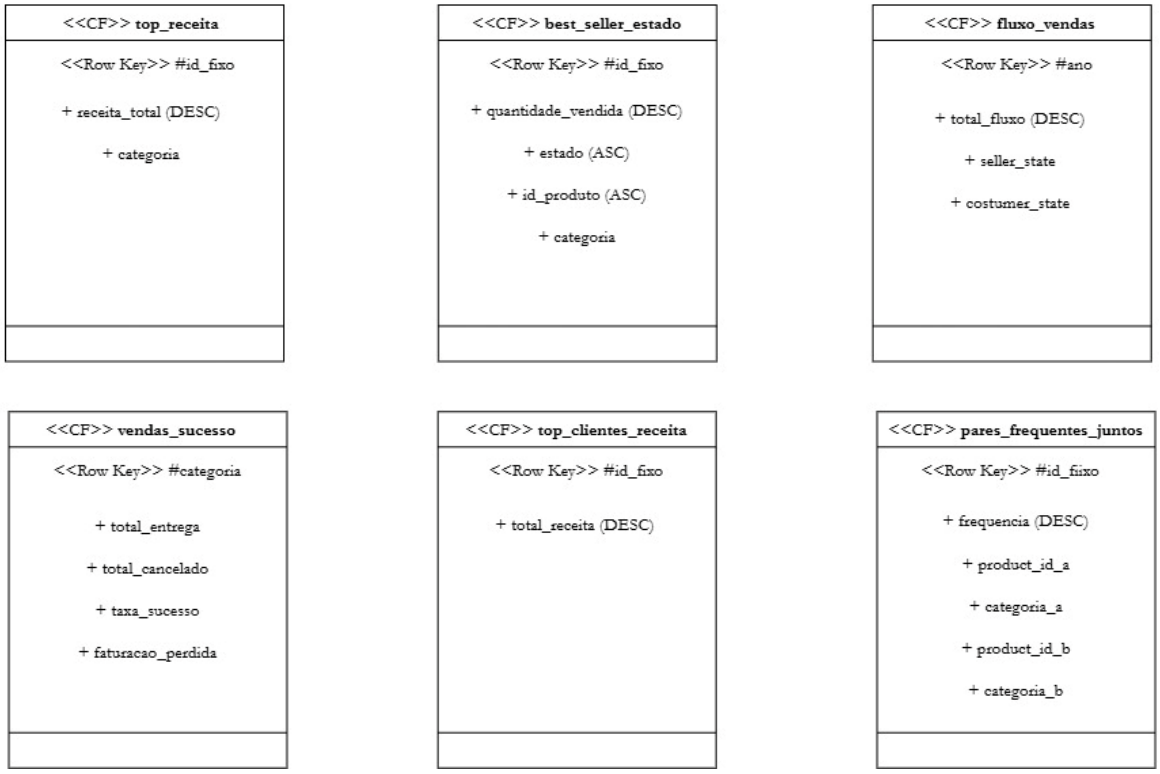

Figura 3 – Modelo de Dados para M2

Questões	Tabela (Column Family)	Partion Key	Clustering Columns	Justificação do Desenho da Chave
Questão 1	top_receita	id_fixo	receita_total (DESC)	A chave estática agrupa todas as categorias na mesma partição global. A Clustering Column garante a ordenação descendente da receita no disco, permitindo a extração imediata do Top 10.
Questão 2	best_seller_estado	id_fixo	quantidade_vendida (DESC) estado (ASC) id_produto (ASC)	A chave de partição estática agrupa todos os estados numa partição global, o que permite extrair os dados do país inteiro numa única leitura. A Clustering Column principal, garante a ordenação descendente pela quantidade vendida (colocando o "Best Seller" no topo), enquanto as restantes chaves asseguram a exclusividade de cada registo.
Questão 3	fluxo_vendas	ano	total_fluxo(DESC)	A partition key permite filtrar os dados pelo ano pretendido. A clustering column ordena as rotas mais frequentes.
Questão 4	sucesso_vendas	categoria	Nenhum	Uma linha por categoria, com os valores já agregados, permitindo acesso direto e eficiente aos resultados.
Questão 5	top_clientes_receita	id_fixo	total_receita (DESC)	Top 10 clientes, ordenado pelo seu volume financeiro gerado.
Questão 6	pares_frequentes_juntos	id_fixo	frequencia (DESC)	Partição estática para alojar todos os pares. A ordenação pela frequência descendente revela os padrões de *cross-selling* mais comuns numa única leitura.

Tabela 18 – Definição das Chaves de Partição e Clustering para o modelo em Cassandra.

3.2.2. Questões Analíticas para Implementação

Questão 1 - Quais são as 10 categorias de produtos que geram maior receita total (preço + taxa de envio) para a plataforma?

```
USE bd2ks;

SELECT categoria, receita_total
FROM top_receita
WHERE id_fixo = 'Geral'
LIMIT 10;
```

categoria	receita_total
beleza_saude	1441248.07
relogios_presentes	1305541.61
cama_mesa_banho	1241681.72
esporte_lazer	1156656.48
informatica_acessorios	1059272.40
moveis_decoracao	902511.79
utilidades_domesticas	778397.77
cool_stuff	719329.95
automotivo	685384.32
ferramentas_jardim	584219.21

Tabela 19 – Top 10 categorias de produtos com maior receita total acumulada (Apache Cassandra)

Questão 2 - Qual é o produto mais vendido ("Best Seller") em termos de quantidade em cada estado do Brasil?

```
USE bd2ks;

SELECT
    estado,
    id_produto,
    categoria,
    quantidade_vendida
FROM best_seller_estado
WHERE id_fixo = 'Geral'
LIMIT 6;
```

estado	id_produto	categoria	quantidade_vendida
SP	aca2eb7d00ea1a7b8ebd4e68314663af	moveis_decoracao	265
RJ	d1c427060a0f73f6b889a5c7c61f2ac4	informatica_acessorios	102
MG	d1c427060a0f73f6b889a5c7c61f2ac4	informatica_acessorios	82
RS	422879e10f46682990de24d770e7f83d	ferramentas_jardim	42
PR	a62e25e09e05e6faf31d90c6ec1aa3d1	relogios_presentes	28
SC	368c6c730842d78016ad823897a372db	ferramentas_jardim	27

Tabela 20 – Identificação do produto Best Seller por estado (Apache Cassandra)

Questão 3 - Qual é o fluxo de vendas entre diferentes estados (Vendedor vs. Cliente) para identificar as rotas comerciais mais frequentes?

```
USE p3;

SELECT *
FROM fluxo_vendas
WHERE ano='2016'
LIMIT 10;
```

ano	total_fluxo	seller_state	customer_state
2016	69	SP	SP
2016	22	SP	MG
2016	21	SP	RJ
2016	15	RJ	RJ
2016	13	PR	SP
2016	12	RJ	SP
2016	12	SP	RS
2016	9	SP	PR
2016	7	PR	PR
2016	6	MG	MG

Tabela 21 – Fluxo comercial entre estados (Apache Cassandra)

Questão 4 - Qual é a taxa de sucesso das vendas, comparando encomendas entregues e canceladas, por categoria de produto e faturação perdida?

```
USE bd2_vendas;

SELECT *
FROM sucesso_vendas;
```

categoria	total_entregue	total_cancelado	taxa_sucesso	faturacao_perdida
beleza_saude	9422	34	99.64	3397.85
sinalizacao_e_seguranca	197	0	100.00	0.00
alimentos	498	1	99.80	167.66
construcao_ferramentas_construcao	916	2	99.78	148.31
livros_tecnicos	263	0	100.00	0.00
livros_interesse_geral	537	7	98.71	1393.71
moveis_escritorio	1662	1	99.94	459.63
telefonia	4422	16	99.64	1250.05
ferramentas_jardim	4263	19	99.56	4694.01
telefonia_fixa	250	3	98.81	1806.76
artigos_de_natal	150	1	99.34	20.93
construcao_ferramentas_jardim	232	4	98.31	245.40
informatica_acessorios	7632	46	99.40	8330.82
fashion_roupa_infanto_juvenil	7	0	100.00	0.00
tablets_impressao_imagem	83	0	100.00	0.00
market_place	294	1	99.66	140.00
audio	360	1	99.72	27.68
utilidades_domesticas	6783	49	99.28	6362.63
construcao_ferramentas_ferramentas	103	0	100.00	0.00
portateis_cozinha_e_preparadores_de_alimentos	14	1	93.33	50.24
agro_industria_e_comercio	206	0	100.00	0.00
cds_dvds_musicais	14	0	100.00	0.00
fashion_bolsas_e_acessorios	1976	5	99.75	1185.27
climatizacao	281	2	99.29	337.62
artes	197	1	99.49	51.60
Sem_Categoria	1535	14	99.10	1180.80

Tabela 22 – Análise da taxa de sucesso de vendas e faturação perdida por categoria (Apache Cassandra)

Questão 5 - Quais são os 10 clientes que geram maior receita total (preço + taxa de envio) para a plataforma?

```
USE bd2_receita;

SELECT
       customer_unique_id,
       total_receita,
FROM top_clientes_receita
WHERE id_fixo = 'Geral'
LIMIT 10;
```

customer_unique_id	total_receita
0a0a92112bd4c708ca5fde585afaa872	13664.08
da122df9eeddfedc1dc1f5349a1a690c	7571.63
763c8b1c9c68a0229c42c9fc6f662b93	7274.88
dc4802a71eae9be1dd28f5d788ceb526	6929.31
459bef486812aa25204be022145caa62	6922.21
ff4159b92c40ebe40454e3e6a7c35ed6	6726.66
4007669dec559734d6f53e029e360987	6081.54
eebb5dda148d3893cdaf5b5ca3040ccb	4764.34
48e1ac109decbb87765a3eade6854098	4681.78
c8460e4251689ba205045f3ea17884a1	4655.88

Tabela 23 – Ranking dos 10 clientes com maior volume financeiro gerado (Apache Cassandra)

Questão 6 - Quais os produtos que são frequentemente comprados em conjunto por categoria na mesma encomenda?

```
USE bd2ks;

SELECT
    product_id_a,
```

```
    categoria_a,
    product_id_b,
    categoria_b,
    frequencia
FROM pares_frequentes_juntos
WHERE id_fixo = 'Geral'
LIMIT 10;
```

product_id_a	categoria_a	product_id_b	categoria_b	frequencia
05b515fdc76e888aada3c6d66c201dff	beleza_saude	270516a3f41dc035aa87d220228f844c	beleza_saude	100
36f60d45225e60c7da4558b070ce4b60	informatica_acessorios	e53e557d5a159f5aa2c5e995dfdf244b	informatica_acessorios	48
62995b7e571f5760017991632bbfd311	utilidades_domesticas	ac1ad58efc1ebf66bfadc09f29bdedc0	utilidades_domesticas	36
710b7c26b7a742f497bba45fab91a25f	utilidades_domesticas	a9d9db064d4afd4458eb3e139fe29167	utilidades_domesticas	36
2ef36e1cae01b86d0ff0a2f50ff2bd53	moveis_decoracao	53759a2ecddad2bb87a079a1f1519f73	ferramentas_jardim	30
308e4e21ae228a10f6370a243ae59995	moveis_escritorio	90b58782fdd04cb829667fcc41fb65f5	moveis_escritorio	30
35afc973633aaeb6b877ff57b2793310	casa_conforto	99a4788cb24856965c36a24e339b6058	cama_mesa_banho	30
18486698933fbb64af6c0a255f7dd64c	informatica_acessorios	dbb67791e405873b259e4656bf971246	informatica_acessorios	26
58efb9b638561ce132216a9a612513e2	moveis_sala	872db866d615db59612ac933f43d6b22	moveis_sala	25
0db4869ac8a908414823d44d742f5ba3	moveis_escritorio	47920da896bbd89e851429c78a7a7a8b	moveis_escritorio	24

Tabela 24 – Listagem dos pares de produtos mais vendidos em conjunto, detalhados por identificador único (ID) e nome da categoria (Apache Cassandra)

3.2.3. Implementação da Base de Dados

A implementação da base de dados em Apache Cassandra exigiu uma abordagem estruturada em três fases principais. Uma vez que esta tecnologia de Column Store é otimizada para escalabilidade e disponibilidade, não suportando operações complexas como junções (JOINs) ou agregações nativas avançadas (ad hoc aggregations), foi necessário pré-processar a informação.

De acordo com as diretrizes metodológicas deste relatório, os extratos de código aqui apresentados servem apenas de ilustração das etapas, encontrando-se a listagem integral do código de cada questão no **Anexo M2 – Cassandra**.

1. Pré-processamento e Exportação de Dados (MySQL): Para contornar as restrições de agregação do Cassandra, os dados de origem foram previamente calculados e agrupados no MySQL. Posteriormente, os resultados foram exportados para ficheiros CSV através da instrução INTO OUTFILE.

Por exemplo, na **Questão 1** como o Cassandra não faz agregações (não soma receitas) nem ordena dados globalmente de forma eficiente no momento da leitura, tivemos de pré-agregar os dados no MySQL. Para a Partition Key (id_fixo = 'Geral') escolhemos uma chave estática ('Geral') porque queriamos um ranking global. Ao usar o mesmo valor para todas as linhas, obrigamos o Cassandra a guardar todas as categorias na mesma partição (no mesmo nó). Para a escolha da Clustering Column (receita_total DESC) definimos a receita total como chave de ordenação descendente, assim o Cassandra guarda os dados fisicamente ordenados do maior para o menor logo no momento da escrita. Assim, a leitura é instantânea: basta ir à partição 'Geral' e pedir as 10 primeiras linhas (LIMIT 10):

```
SELECT
    'Geral' AS filtro,
    CASE
        WHEN p.product_category_name IS NULL OR p.product_category_name = '' THEN 'Sem_Categoria'
        ELSE p.product_category_name
    END AS categoria,
    SUM(oi.price + oi.freight_value) AS receita_total
(…)

INTO OUTFILE '/var/lib/mysql/csv/pergunta1.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

Na **Questão 2**, o objetivo consistia em identificar o produto mais vendido ("Best Seller") em termos de quantidade para cada estado do Brasil.

Optou-se por utilizar uma Partition Key estática (id_fixo = 'Geral') para agrupar todos os estados na mesma partição. Esta abordagem permite extrair a tabela global do país inteiro através de uma única consulta de leitura no Cassandra.

A complexidade analítica de descobrir qual o produto com o valor máximo de vendas em cada região foi totalmente resolvido durante a fase de pré-processamento no MySQL. Para tal, utilizou-se uma subquery na cláusula HAVING que filtrou os dados antes da exportação, garantindo assim que o ficheiro CSV gerasse apenas uma linha por estado.

Para as Clustering Columns, foi definida a ordenação principal por quantidade_vendida DESC, seguida pelo estado e id_produto para assegurar a unicidade de cada registo na Primary Key.

Deste modo, o Cassandra apenas necessita de ler a partição e devolver os resultados já fisicamente ordenados no disco.

```
SELECT
    'Geral' AS id_fixo,
    c.customer_state AS estado,
    oi.product_id AS id_produto,
    CASE
        WHEN p.product_category_name IS NULL OR p.product_category_name = '' THEN 'Sem_Categoria'
        ELSE p.product_category_name
    END AS categoria,
    COUNT(oi.product_id) AS quantidade_vendida
FROM Customer c
JOIN `Order` o ON c.customer_id = o.customer_id
JOIN Order_item oi ON o.order_id = oi.order_id
JOIN Product p ON oi.product_id = p.product_id
GROUP BY c.customer_state, oi.product_id, p.product_category_name
HAVING (c.customer_state, COUNT(oi.product_id)) IN (
    -- Subquery para encontrar o máximo vendido por estado
    SELECT estado_sub, MAX(total_vendas)
    FROM (
        SELECT c2.customer_state AS estado_sub, COUNT(oi2.product_id) AS total_vendas
        FROM Customer c2
        JOIN `Order` o2 ON c2.customer_id = o2.customer_id
        JOIN Order_item oi2 ON o2.order_id = oi2.order_id
        GROUP BY c2.customer_state, oi2.product_id
    ) AS sub_tabela
    GROUP BY estado_sub
)
ORDER BY quantidade_vendida DESC
INTO OUTFILE '/var/lib/mysql/csv/questao2.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

Na **Questão 3**, o objetivo consistiu em identificar o fluxo de vendas entre diferentes estados, considerando o estado do vendedor como origem e o estado do cliente como destino. À semelhança da questão 1, foi necessário realizar a exportação dos dados em MySQL.

Foi necessário combinar informação proveniente de várias tabelas (Customer, Order, Order_item e Seller), de forma a associar cada encomenda ao respetivo estado de origem e destino. Posteriormente, os dados foram agregados por ano e por par de estados (vendedor vs. cliente), contabilizando o número de encomendas distintas, que representa o fluxo comercial.

O cálculo foi realizado através da seguinte query em MySQL:

```
SELECT
    YEAR(o.order_purchase_timestamp) AS ano,
    c.customer_state,
    s.seller_state,
    COUNT(DISTINCT o.order_id) AS total_fluxo
FROM Customer c
JOIN `Order` o ON c.customer_id = o.customer_id
JOIN Order_item oi ON o.order_id = oi.order_id
JOIN Seller s ON oi.seller_id = s.seller_id
GROUP BY ano, s.seller_state, c.customer_state
INTO OUTFILE '/var/lib/mysql/csv/pergunta3.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

Tal como nas restantes questões, os resultados foram exportados para um ficheiro CSV, que posteriormente foi utilizado para carregamento no Cassandra.

Relativamente ao modelo de dados em Cassandra, foi necessário conceber a tabela de acordo com o padrão de acesso pretendido. Foi definida:

Partition Key: ano, permitindo distribuir os dados por diferentes nós e consultar eficientemente os fluxos de um determinado ano;

Clustering Columns: seller_state e customer_state, a organizar os dados dentro de cada partição por origem e destino;

Coluna de métrica: total_fluxo, previamente calculada.

Na **Questão 4**, para contornar as restrições de agregação do Cassandra, os dados foram previamente calculados e agrupados no MySQL, sendo posteriormente exportados para ficheiros CSV através da instrução INTO OUTFILE.

Como o Cassandra não suporta contagens condicionais nem cálculos de percentagens em tempo de leitura, foi necessário pré-agregar os dados por categoria de produto, obtendo o número de encomendas entregues, canceladas, a taxa de sucesso e a faturação perdida.

Ao contrário da Questão 1, por exemplo, não foi necessário utilizar uma Partition Key estática, uma vez que a análise é feita por categoria e não em formato de ranking. Para garantir a consistência dos dados, , foi necessário tratar categorias nulas durante o pré-processamento.

Desta forma, os dados ficam preparados no MySQL no formato final, permitindo a sua importação direta para o Cassandra sem necessidade de processamento adicional.

```
SELECT
    CASE
        WHEN p.product_category_name IS NULL OR p.product_category_name = '' THEN 'Sem_Categoria'
        ELSE p.product_category_name
    END AS categoria,
    SUM(IF(o.order_status = 'delivered', 1, 0)) AS total_entregue,
    SUM(IF(o.order_status = 'canceled', 1, 0)) AS total_cancelado,
    ROUND(
        SUM(IF(o.order_status = 'delivered', 1, 0)) /
        NULLIF(SUM(IF(o.order_status IN ('delivered','canceled'), 1, 0)), 0) * 100,
        2
    ) AS taxa_sucesso,
    SUM(IF(o.order_status = 'canceled', oi.price + oi.freight_value, 0)) AS faturacao_perdida
FROM Product p
JOIN Order_item oi ON p.product_id = oi.product_id
JOIN `Order` o ON oi.order_id = o.order_id
GROUP BY p.product_category_name
INTO OUTFILE '/var/lib/mysql/csv/pergunta4.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

Na **Questão 5** foi seguida a mesma abordagem geral descrita na Questão 1 relativamente ao pré-processamento dos dados em MySQL.

Neste caso, a agregação foi realizada ao nível do cliente, sendo calculada a receita total por customer_unique_id através da soma dos atributos price e freight_value, considerando apenas encomendas com estado 'delivered'. Desta forma, cada linha do resultado corresponde a um cliente com a respetiva receita total já calculada. Adicionalmente, foi incluído um identificador fixo ('Geral' AS id_fixo) no resultado, com o objetivo de permitir a criação de um ranking global de clientes na fase de armazenamento em Cassandra.

O resultado final foi ordenado de forma decrescente pela receita total e exportado para um ficheiro CSV, sendo posteriormente utilizado para carregamento direto numa estrutura otimizada para leitura no Cassandra.

```
SELECT
    'Geral' AS id_fixo,
    c.customer_unique_id,
    SUM(oi.price + oi.freight_value) AS total_receita
FROM `Order` o
JOIN customer c ON c.customer_id = o.customer_id
JOIN Order_item oi ON oi.order_id = o.order_id
```

```
WHERE o.order_status = 'delivered'
GROUP BY c.customer_unique_id
ORDER BY total_receita DESC
INTO OUTFILE '/var/lib/mysql/csv/pergunta5.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

Na **Questão 6**, o objetivo consistia em identificar os pares de produtos que são frequentemente comprados em conjunto na mesma encomenda.

Optou-se por utilizar uma Partition Key estática (id_fixo = 'Geral') para agrupar todas as combinações de produtos numa partição única. Esta abordagem permite obter o ranking global dos pares mais vendidos no país inteiro através de uma leitura direta e simples.

A complexidade analítica de cruzar e contabilizar os produtos comprados em conjunto foi totalmente resolvida durante a fase de pré-processamento no MySQL.

Para tal, utilizou-se um auto-relacionamento (self-join) da tabela de itens de encomenda, e foi aplicada a restrição oi1.product_id < oi2.product_id.

Esta lógica foi crucial para garantir que os pares não são contabilizados em duplicado (evitando cruzamentos do tipo A-B e B-A) e que o mesmo produto não é cruzado consigo próprio.

Para as Clustering Columns, foi definida a ordenação principal através da métrica de frequência DESC, seguida pelas chaves product_id_a e product_id_b para assegurar a exclusividade de cada registo na Primary Key, evitando assim a perda de dados em caso de empate na frequência.

Desta forma, os pares com maior afinidade de compra ficam fisicamente armazenados no topo da estrutura, permitindo ao Cassandra devolver os resultados (LIMIT 10) de forma instantânea.

```
DROP KEYSPACE IF EXISTS bd2ks;

CREATE KEYSPACE bd2ks WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1};

USE bd2ks;

CREATE TABLE pares_frequentes_juntos (
       id_fixo text,
       frequencia int,
    product_id_a text,
    categoria_a text,
    product_id_b text,
    categoria_b text,
    PRIMARY KEY (id_fixo, frequencia, product_id_a, product_id_b)
) WITH CLUSTERING ORDER BY (frequencia DESC);
```

2. Criação do Keyspace e Tabelas (Cassandra): O ambiente Cassandra foi acedido via terminal (cqlsh) e gerido através da interface do DBeaver. O processo de implementação iniciou-se com a criação do Keyspace bd2ks, configurado com a estratégia de replicação simples (SimpleStrategy) e um fator de replicação de 1:

```
DROP KEYSPACE IF EXISTS bd2ks;

CREATE KEYSPACE bd2ks WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1};
```

Em seguida, instanciaram-se as tabelas utilizando comandos CREATE TABLE desenhados de acordo com o modelo de dados previamente justificado na secção 3.2.1 e estão todas em anexo.

3. Importação de Dados (cqlsh) Com a estrutura da base de dados definida, o povoamento físico foi executado de forma massiva pela linha de comandos do terminal (cqlsh). Recorreu-se ao comando COPY para injetar os dados dos ficheiros CSV gerados na primeira fase, diretamente para as Column Families correspondentes, por exemplo:

```
USE bd2ks;
COPY top_receita(id_fixo, categoria, receita_total)
FROM '/var/lib/cassandra/pergunta1.csv'
```

```
WITH DELIMITER=';';
```

Este fluxo de implementação (extração agregada, modelação Query First e importação de ficheiros) foi replicado com sucesso para as restantes questões analíticas que se encontram documentadas no **Anexo M2 – Cassandra** e cujas consultas de leitura (SELECT) constam da secção 3.2.2..

3.3. Momento 3

3.3.1. Modelo dos Dados para Implementação

Para a implementação em Neo4j (M3), a transformação do modelo relacional exigiu uma nova mudança de paradigma para os Graph Database Systems (Sistemas de Bases de Dados em Grafo). Ao contrário do modelo relacional assente em tabelas normalizadas, ou do modelo do Cassandra focado na desnormalização orientada a consultas, o Neo4j permite representar os dados da forma mais próxima da realidade do domínio: através de entidades (Nós) e das suas ligações explícitas (Relações).

Para uma perceção de alto nível da topologia do grafo implementado, a interface nativa da base de dados (Neo4j Browser, acessível via http://localhost:7474) permite extrair automaticamente o esquema estrutural através do comando CALL db.schema.visualization();.

A Figura 4 ilustra o resultado desta execução, evidenciando as ligações direcionadas entre as entidades principais do domínio.

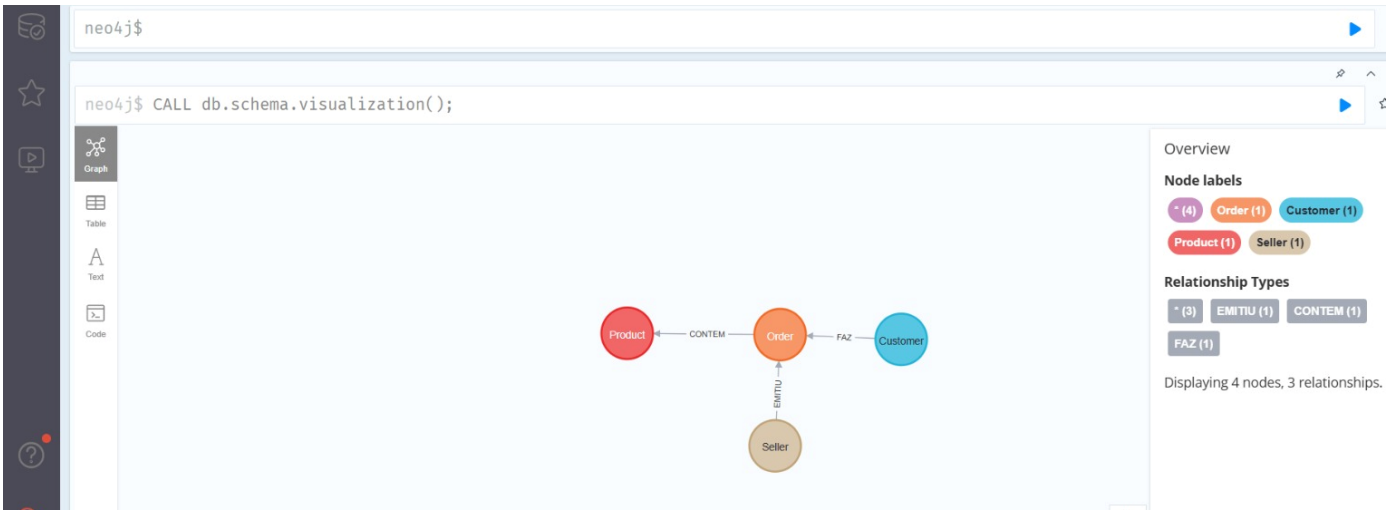

Figura 4 – Representação automática do esquema lógico do grafo no Neo4j Browser

Apesar da representação automática fornecer uma visão global da topologia, a modelação física requer o mapeamento detalhado das propriedades. De seguida (Figura 5), ilustra-se o modelo de dados completo desenhado pelo grupo, à semelhança das representações concetuais de grafos.

Primeiramente pensamos em colocar adicionalmente a entidade :Order_item como nó, posteriormente essa ideia foi descartada de maneira a otimizar os nossos saltos entre nós colocando os atributos da mesma como propriedades na ligação [:CONTEM].

À semelhança, o modelo final estruturou-se em duas componentes essenciais:

- **Nós (Nodes):** Representam as entidades fundamentais do negócio (:Customer, :Order, :Seller, e :Product), contendo os seus atributos intrínsecos como propriedades.
- **Relações (Relationships):** Substituem as tradicionais chaves estrangeiras e tabelas associativas do MySQL. Destaca-se a grande otimização efetuada na transição da antiga tabela Order_item. No grafo, esta entidade deixou de existir como um Nó, sendo absorvida pela relação direcionada [:CONTEM], que passou a armazenar ela própria os metadados transacionais (como o price e o freight_value).

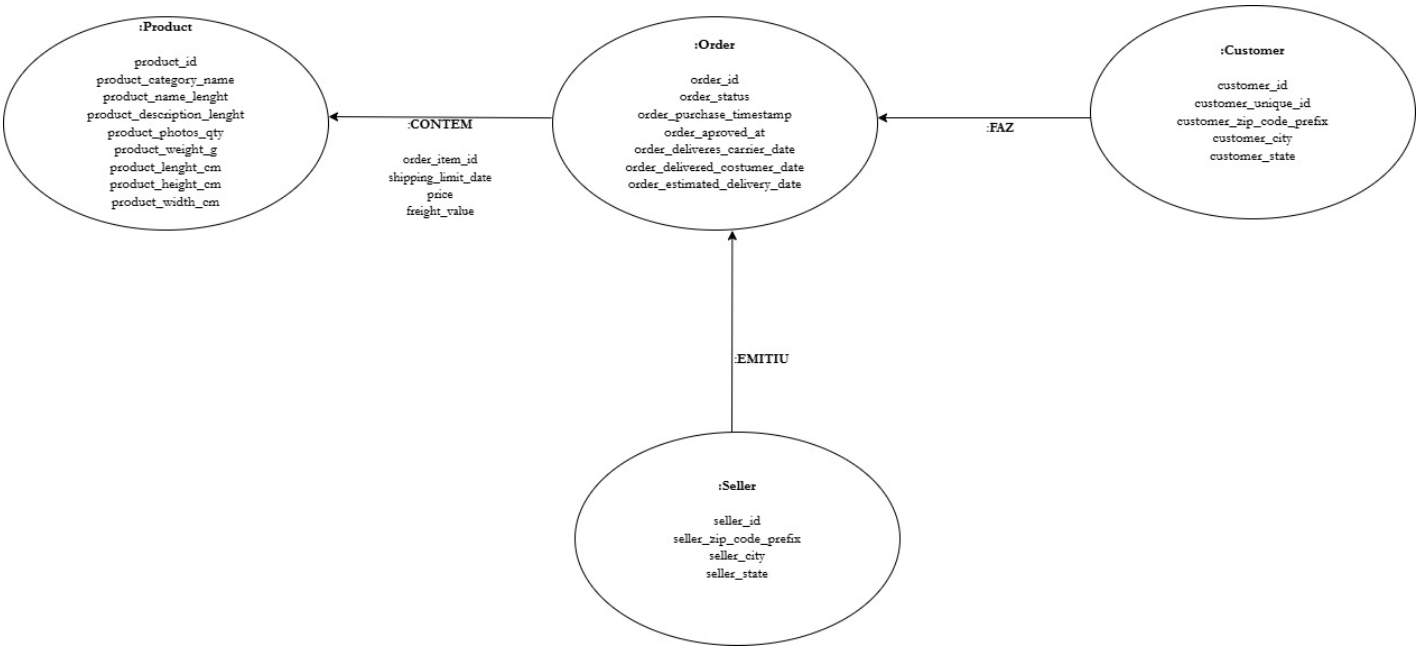

Figura 5 – Modelo de Dados Detalhado para M3 (Neo4j)

3.3.2. Questões Analíticas para Implementação

Questão 1 - Quais são as 10 categorias de produtos que geram maior receita total (preço + taxa de envio) para a plataforma?

```
MATCH (o:Order)-[c:CONTEM]->(p:Product)
RETURN
    CASE
        WHEN p.product_category_name IS NULL OR
p.product_category_name = '' THEN 'Sem_Categoria'
        ELSE p.product_category_name
    END AS categoria,
    sum(c.price + c.freight_value) AS receita_total ORDER BY
receita_total DESC
LIMIT 10;
```

	categoria	receita_total
1	beleza_saude	1.441.248,0700000087
2	relogios_presentes	1.305.541,6100000106
3	cama_mesa_banho	1.241.681,7200000035
4	esporte_lazer	1.156.656,4800000112
5	informatica_acessorios	1.059.272,4000000001
6	moveis_decoracao	902.511,7900000038
7	utilidades_domesticas	778.397,7700000016
8	cool_stuff	719.329,950000003
9	automotivo	685.384,3199999981
10	ferramentas_jardim	584.219,2099999995

Tabela 25 – Top 10 categoria de produtos com maior receita total acumula (Cypher)

Questão 2 - Qual é o produto mais vendido ("Best Seller") em termos de quantidade em cada estado do Brasil?

```
MATCH (c:Customer)-[:FAZ]->(o:Order)-[:CONTEM]->(p:Product)

WITH c.customer_state AS estado,
     p.product_id AS id_produto,
     p.product_category_name AS categoria,
     count(p) AS quantidade_vendida

ORDER BY quantidade_vendida DESC

WITH estado, collect({
    id: id_produto,
    categoria: categoria,
    quantidade: quantidade_vendida
})[0] AS best_seller

RETURN estado AS Estado,
       best_seller.id AS ID_Produto,
       best_seller.categoria AS Categoria,
       best_seller.quantidade AS Quantidade_Vendida
```

	Estado	ID_Produto	Categoria	Quantidade_Vendida
1	SP	aca2eb7d00ea1a7b8ebd4e68314663af	moveis_decoracao	265
2	RJ	d1c427060a0f73f6b889a5c7c61f2ac4	informatica_acessorios	102
3	MG	d1c427060a0f73f6b889a5c7c61f2ac4	informatica_acessorios	82
4	RS	422879e10f46682990de24d770e7f83d	ferramentas_jardim	42
5	PR	a62e25e09e05e6faf31d90c6ec1aa3d1	relogios_presentes	28
6	SC	368c6c730842d78016ad823897a372db	ferramentas_jardim	27

Tabela 26 – Identificação do produto Best Seller por estado (Cypher)

```
ORDER BY Quantidade_Vendida DESC;
```

Questão 3. Qual é o fluxo de vendas entre diferentes estados (Vendedor vs. Cliente) para identificar as rotas comerciais mais frequentes?

```
MATCH (c:Customer)-[:FAZ]->(o:Order)<-[:EMITIU]-(s:Seller)
WHERE date(o.order_purchase_timestamp).year = 2016

RETURN c.customer_state AS customer_state, s.seller_state
AS seller_state, count(DISTINCT o.order_id) AS total_fluxo

ORDER BY total_fluxo DESC;
```

	customer_state	seller_state	total_fluxo
1	SP	SP	69
2	MG	SP	22
3	RJ	SP	21
4	RJ	RJ	15
5	SP	PR	13
6	RS	SP	12
7	SP	RJ	12
8	PR	SP	9
9	PR	PR	7
10	MG	MG	6
11	SC	SP	6
12	RJ	PR	5

Tabela 27 – Fluxo comercial entre estados (Cypher)

Questão 4 - Qual é a taxa de sucesso das vendas, comparando encomendas entregues e canceladas, por categoria de produto e faturação perdida?

```
MATCH (o:Order)-[c:CONTEM]->(p:Product)
WHERE o.order_status IN ['delivered', 'canceled']

  AND o.order_purchase_timestamp.year IN [2017, 2018]

WITH
  CASE
    WHEN p.product_category_name IS NULL OR p.product_category_name = '' THEN 'Sem Categoria'
    ELSE p.product_category_name
  END AS categoria,
  o.order_status AS status,
  c.price + c.freight_value AS valor_item

WITH categoria,
     sum(CASE WHEN status = 'delivered' THEN 1 ELSE 0 END) AS total_entregue,
     sum(CASE WHEN status = 'canceled' THEN 1 ELSE 0 END) AS total_cancelado,
     sum(CASE WHEN status = 'canceled' THEN valor_item ELSE 0 END) AS faturacao_perdida

RETURN categoria,
       total_entregue,
       total_cancelado,
       CASE
         WHEN (total_entregue + total_cancelado) = 0 THEN 0.0
         ELSE (toFloat(total_entregue) / (total_entregue + total_cancelado)) * 100
       END AS taxa_sucesso,
       faturacao_perdida

ORDER BY faturacao_perdida DESC;
```

	categoria	total_entregue	total_cancelado	taxa_sucesso	faturacao_perdida
1	cool_stuff	3.711	16	99,5707002951	15.153,48
2	informatica_acessorios	7.632	46	99,4008856473	8.330,82
3	esporte_lazer	8.414	49	99,4210090984	7.975,5
4	relogios_presentes	5.855	21	99,6426140231	7.249,14
5	utilidades_domesticas	6.783	49	99,2827868852	6.362,63
6	automotivo	4.132	28	99,3269230769	6.272,58
7	ferramentas_jardim	4.263	19	99,5562821112	4.694,01
8	brinquedos	4.006	31	99,2321030468	4.194,71
9	instrumentos_musicais	651	11	98,3383685801	4.116,05
10	beleza_saude	9.422	34	99,6404399323	3.397,85

Tabela 28 – Análise da taxa de sucesso de vendas e faturação perdida por categoria (Cypher)

Questão 5 - Quais são os 10 clientes que geram maior receita total (preço + taxa de envio) para a plataforma?

```
MATCH (c:Customer)-[:FAZ]->(o:Order)-[ct:CONTEM]->(p:Product)
```

```
WITH c.customer_id AS cliente,
     c.customer_city AS cidade,
     c.customer_state AS estado,
     sum(ct.price + ct.freight_value) AS
receita_total

RETURN cliente,
       cidade,
       estado,
       receita_total

ORDER BY receita_total DESC
LIMIT 10;
```

	cliente	cidade	estado	receita_total
1	1617b1357756262bfa56ab541c47bc16	rio de janeiro	RJ	122,976.7199999998
2	ec5b2ba62e574342386871631fafd3fc	vila velha	ES	65,473.92
3	c6e2731c5b391845f6800c97401a43a9	campo grande	MS	62,363.79
4	f48d464a0baaea338cb25f816991ab1f	vitoria	ES	62,299.89
5	3fd6777bbce08a352fddd04e4a7cc8f6	marilia	SP	60,539.94
6	05455dfa7cd02f13d132aa7a6a9729c6	divinopolis	MG	54,733.8599999999
7	df55c14d1476a9a3467f131269c2477f	araruama	RJ	44,553.06
8	e0a2412720e9ea4f26c1ac985f6a7358	goiania	GO	43,284.96
9	24bbf5fd2f2e1b359ee7de94defc4a15	maua	SP	42,879.06
10	3d979689f636322c62418b6346b1c6d2	joao pessoa	PB	42,136.02

Tabela 29 – Ranking dos 10 clientes com maior volume financeiro gerado (Cypher)

Questão 6 - Quais os produtos que são frequentemente comprados em conjunto por categoria na mesma encomenda?

```
MATCH (o:Order)-[:CONTEM]->(p1:Product), (o)-[:CONTEM]->(p2:Product)

WHERE p1.product_id < p2.product_id

RETURN
    p1.product_id AS produto_A,
    p1.product_category_name AS categoria_A,
    p2.product_id AS produto_B,
    p2.product_category_name AS categoria_B,
    COUNT(*) AS frequencia

ORDER BY frequencia DESC
LIMIT 10;
```

	produto_A	categoria_A	produto_B	categoria_B	frequencia
1	05b515fdc76e888aada3c6d66c201dff	beleza_saude	270516a3f41dc035aa87d220228f844c	beleza_saude	100
2	36f60d45225e60c7da4558b070ce4b60	informatica_acessorios	e53e557d5a159f5aa2c5e995dfdf244b	informatica_acessorios	48
3	710b7c26b7a742f497bba45fab91a25f	utilidades_domesticas	a9d9db064d4afd4458eb3e139fe29167	utilidades_domesticas	36
4	62995b7e571f5760017991632bbfd311	utilidades_domesticas	ac1ad58efc1ebf66bfadc09f29bdedc0	utilidades_domesticas	36
5	2ef36e1cae01b86d0ff0a2f50ff2bd53	moveis_decoracao	53759a2ecddad2bb87a079a1f1519f73	ferramentas_jardim	30
6	35afc973633aaeb6b877ff57b2793310	casa_conforto	99a4788cb24856965c36a24e339b6058	cama_mesa_banho	30
7	308e4e21ae228a10f6370a243ae59995	moveis_escritorio	90b58782fdd04cb829667fcc41fb65f5	moveis_escritorio	30
8	18486698933fbb64af6c0a255f7dd64c	informatica_acessorios	dbb67791e405873b259e4656bf971246	informatica_acessorios	26
9	58efb9b638561ce132216a9a612513e2	moveis_sala	872db866d615db59612ac933f43d6b22	moveis_sala	25
10	ad4b5def91ac7c575dbdf65b5be311f4	informatica_acessorios	e6b314a2236c162ede1a879f1075430f	informatica_acessorios	24

Tabela 30 – Listagem dos pares de produtos mais vendidos em conjunto, detalhados por identificador único (ID) e nome da categoria (Cypher)

3.3.3. Implementação da Base de Dados

A implementação da base de dados em Neo4j constituiu o terceiro momento do projeto e marcou a introdução do paradigma de grafos. Ao contrário do modelo relacional (normalizado) e do Cassandra (desnormalizado e orientado à consulta), o Neo4j privilegia a expressividade das ligações entre entidades, representando-as através de nós e relações que modelam diretamente as interações do domínio.

1. Definição de Restrições e Integridade

Durante a importação de dados no Neo4j, a ausência de restrições obrigava o sistema a realizar Full Node Scans demorados, bloqueando a criação das relações. Para resolver o problema, aplicámos o comando CREATE CONSTRAINT nos identificadores principais (customer_id, order_id, seller_id, product_id) antes do carregamento. Esta solução garantiu a absoluta integridade do grafo, evitando nós duplicados, e gerou índices automáticos para esses atributos. Graças a esta indexação, as pesquisas tornaram-se instantâneas, reduzindo o tempo de ligação entre os nós de várias horas para apenas alguns segundos.

```
CREATE CONSTRAINT FOR (c:Customer) REQUIRE c.customer_id IS UNIQUE;
CREATE CONSTRAINT FOR (o:Order) REQUIRE o.order_id IS UNIQUE;
CREATE CONSTRAINT FOR (s:Seller) REQUIRE s.seller_id IS UNIQUE;
CREATE CONSTRAINT FOR (p:Product) REQUIRE p.product_id IS UNIQUE;
```

2. Importação de Dados e Criação de Nós (Nodes)

Após a definição das restrições de unicidade, procedeu-se à importação física dos dados a partir dos ficheiros CSV gerados e preparados no momento relacional. Com a transição para o paradigma de grafos, o conceito tradicional de "tabelas" desaparece, dando lugar à criação de Nós (Nodes), que representam as entidades principais do domínio (como Customer, Seller e Product).

O processo foi executado através do comando LOAD CSV WITH HEADERS, que iterou sobre os ficheiros para instanciar cada nó com a sua respetiva etiqueta (Label) e propriedades. O exemplo seguinte demonstra a criação simples e direta dos nós referentes aos clientes:

```
LOAD CSV WITH HEADERS
FROM 'file:///olist_customers_dataset.csv' AS csvLineCustomer

CREATE (customer:Customer {
    customer_id: csvLineCustomer.customer_id,
    customer_unique_id: csvLineCustomer.customer_unique_id,
    customer_zip_code_prefix: csvLineCustomer.customer_zip_code_prefix,
    customer_city: csvLineCustomer.customer_city,
    customer_state: csvLineCustomer.customer_state
});
```

Um aspeto técnico de relevo nesta fase de importação prende-se com a natureza schema-free do Neo4j. Ao contrário da rigidez imposta pelas chaves de partição do Cassandra (que rejeitava importações com categorias nulas), a criação dos nós referentes aos Produtos (:Product) processou-se sem qualquer erro de inserção perante atributos vazios, demonstrando a elevada flexibilidade estrutural do modelo de grafos.

3. Modelação das Relações (Edges) e Propriedades

A população do grafo foi realizada com o comando LOAD CSV, o que permitiu consumir utilizar os ficheiros originais da Olist, sem necessidade de pré-agregações como no Cassandra. As chaves estrangeiras foram convertidas em relações explícitas:

- **FAZ**: liga cada cliente (Customer) ao respetivo pedido (Order).
- **EMITIU**: associa o vendedor (Seller) à ordem de venda.
- **CONTEM**: conecta o pedido (Order) ao produto (Product).

A maior alteração arquitetónica face ao modelo relacional (MySQL) ocorreu na conversão da tabela associativa Order_item. Em vez de se instanciar esta tabela como um novo Nó, a mesma foi inteiramente absorvida pela relação :CONTEM. O Neo4j permite que as próprias relações possuam atributos, o que possibilitou incorporar metadados transacionais (como o price e o freight_value) diretamente na ligação entre a encomenda e o produto, simplificando o grafo. No **Anexo M3 – Neo4j** tem todos o código documentado, mas fica a seguir o extrato de código que ilustra o processo de correspondência dos nós e a respetiva criação das relações descritas:

```
LOAD CSV WITH HEADERS
FROM 'file:///olist_order_items_dataset.csv' AS csvLineOrderItem
MATCH (order:Order {order_id: csvLineOrderItem.order_id})
MATCH (product:Product {product_id: csvLineOrderItem.product_id})
MATCH (seller:Seller {seller_id: csvLineOrderItem.seller_id})
CREATE (seller)-[:EMITIU]->(order)
CREATE (order)-[:CONTEM {
    order_item_id: toInteger(csvLineOrderItem.order_item_id),
    shipping_limit_date:datetime(replace(csvLineOrderItem.shipping_limit_date,' ','T')),
    price: toFloat(csvLineOrderItem.price),
    freight_value: toFloat(csvLineOrderItem.freight_value)
}]->(product);
```

4. Exploração Analítica (Consultas em Cypher)

Na **Questão 1**, a tradução para Cypher, baseada no padrão MATCH (o:Order)-[c:CONTEM]->(p:Product), destaca três vantagens arquitetónicas face aos modelos anteriores: A extração de price e freight_value diretamente da ligação [:CONTEM] simplificou e acelerou a travessia no grafo para a resolução da consulta. Além disso, a utilização da função de agregação sum() agrupa automaticamente os resultados, dispensando a declaração explícita de GROUP BY exigida em SQL e tornando a sintaxe mais limpa. Por fim, para garantir a coerência analítica no ambiente *schema-free* do Neo4j, implementou-se um CASE WHEN na cláusula RETURN, agrupando dinamicamente os produtos sem classificação sob a etiqueta 'Sem_Categoria'. Como podemos ver nesta parte da resolução da querie em cypher:

```
(…)
CASE
        WHEN p.product_category_name IS NULL OR p.product_category_name = '' THEN 'Sem_Categoria'
        ELSE p.product_category_name
    END AS categoria
(…)
```

A **Questão 2** exige identificar o produto líder em vendas por estado. No Neo4j, a complexidade é resolvida através da navegação nativa no grafo e do uso de estruturas de listas (collect).

A lógica de implementação baseia-se num percurso de três saltos: (Customer)-[:FAZ]->(Order)-[:CONTEM]->(Product). Após o percurso, os dados são agregados por estado e produto. A diferença aqui é a utilização da ordenação descendente seguida da função collect()[0]. Esta técnica permite agrupar todos os produtos de um estado numa lista já ordenada e extrair apenas o primeiro elemento, que corresponde ao "Best Seller" absoluto daquela região.

```
MATCH (c:Customer)-[:FAZ]->(o:Order)-[:CONTEM]->(p:Product)
WITH c.customer_state AS estado,
     p.product_id AS id_produto,
     p.product_category_name AS categoria,
     count(p) AS quantidade_vendida
ORDER BY quantidade_vendida DESC
WITH estado, collect({
    id: id_produto,
    categoria: categoria,
    quantidade: quantidade_vendida
})[0] AS best_seller
```

Este fluxo demonstra a eficiência do Cypher em reduzir a verbosidade do código face ao SQL, mantendo a capacidade analítica sem a necessidade de pré-processamento externo que foi exigido no Cassandra.

Na **Questão 3**, pretendeu-se identificar o fluxo de vendas entre estados de clientes e vendedores, de forma a perceber quais as rotas comerciais mais frequentes. Em Neo4j, esta análise foi simplificada através da navegação direta no grafo, utilizou-se o caminho (Customer)-[:FAZ]->(Order)<-[:EMITIU]-(Seller).

Além disso, foi aplicado um filtro temporal para restringir a análise às encomendas realizadas no ano de 2016, como se pode observar neste excerto:

```
MATCH (c:Customer)-[:FAZ]->(o:Order)<-[:EMITIU]-(s:Seller)
WHERE date(o.order_purchase_timestamp).year = 2016
```

Posteriormente, utilizou-se count(DISTINCT o.order_id) para contabilizar o número de encomendas entre estados de clientes e vendedores. A agregação é feita automaticamente pelo Cypher com base nos atributos do RETURN, dispensando a utilização explícita de GROUP BY, tornando a query mais simples e legível.

```
RETURN c.customer_state AS customer_state, s.seller_state AS seller_state, count(DISTINCT o.order_id) AS
total_fluxo
ORDER BY total_fluxo DESC;
```

Esta implementação demonstra como o Cypher permite representar relações entre entidades de forma mais intuitiva e menos verbosa, facilitando a identificação das rotas comerciais mais frequentes.

Durante a validação cruzada dos resultados da **Questão 4**, o grupo detetou uma ligeira discrepância no volume de encomendas entregues (total_entregue) entre o modelo relacional e o modelo de grafos (Neo4j). As métricas de cancelamento e faturação perdida cruzam exatamente ao cêntimo em todos os sistemas (comprovando a exatidão algorítmica das queries).
Esta variação nos itens entregues justifica-se por dois fatores arquitetónicos: primeiramente, enquanto o modelo relacional impõe restrições estritas de Chave Primária na tabela associativa (rejeitando linhas duplicadas no *dataset*), o Neo4j permite a criação de múltiplas relações [:CONTEM] paralelas entre os mesmos nós. Adicionalmente, identificou-se que o Neo4j armazena o campo order_purchase_timestamp como um tipo temporal nativo (DateTime com fuso horário), exigindo uma filtragem direta pelo componente .year. Pequenas divergências na forma como cada motor de base de dados processa transições de fuso horário em fusos limítrofes podem isolar ou incluir registos nas franjas dos anos de 2017 e 2018. Esta constatação empírica evidencia as diferenças de rigor estrutural e tipagem na ingestão de dados entre os paradigmas.

```
MATCH (o:Order)-[c:CONTEM]->(p:Product)
WHERE o.order_status IN ['delivered', 'canceled']

  AND o.order_purchase_timestamp.year IN [2017, 2018]
```

Na **Questão 5**, pretendeu-se identificar os clientes que geram maior receita total para a plataforma. Em Neo4j, esta análise foi realizada através da navegação entre clientes, encomendas e produtos, utilizando o percurso (Customer)-[:FAZ]->(Order)-[:CONTEM]->(Product).
A lógica da implementação baseia-se na agregação do valor total gasto por cada cliente, somando o preço dos produtos e os custos de envio presentes na relação [:CONTEM]. Posteriormente, os resultados foram ordenados de forma decrescente, permitindo identificar os clientes com maior contribuição financeira para a plataforma.

```
MATCH (c:Customer)-[:FAZ]->(o:Order)-[ct:CONTEM]->(p:Product)
WITH   c.customer_id AS cliente,
       c.customer_city AS cidade,
       c.customer_state AS estado,
       sum(ct.price + ct.freight_value) AS receita_total
RETURN cliente, cidade, estado, receita_total
ORDER BY receita_total DESC
LIMIT 10;
```

Esta implementação demonstra a simplicidade do Cypher na realização de agregações e exploração de relações entre entidades, permitindo obter resultados analíticos de forma intuitiva e menos verbosa.

Na **Questão 6**, pretendeu-se identificar quais os produtos frequentemente comprados em conjunto na mesma encomenda, analisando simultaneamente as suas categorias. Em Neo4j, esta análise foi implementada através da navegação entre encomendas e produtos, utilizando o padrão (Order)-[:CONTEM]->(Product), permitindo encontrar pares de produtos associados à mesma compra sem necessidade de JOINs complexos.
A lógica da consulta baseia-se na associação de dois produtos (p1 e p2) pertencentes à mesma encomenda (Order). Para evitar pares duplicados, como (A,B) e (B,A), e impedir que um produto seja comparado consigo próprio, foi utilizada uma condição de ordenação sobre os identificadores dos produtos, como demonstrado no seguinte excerto:

```
MATCH (o:Order)-[:CONTEM]->(p1:Product),(o)-[:CONTEM]->(p2:Product)
WHERE p1.product_id < p2.product_id
```

Posteriormente, a query retorna os identificadores e categorias dos produtos, utilizando COUNT(*) para contabilizar quantas vezes cada par aparece em conjunto nas encomendas. Tal permitiu identificar os pares de produtos mais frequentemente adquiridos pelos clientes.

```
RETURN
    p1.product_id AS produto_A,
    p1.product_category_name AS categoria_A,
    p2.product_id AS produto_B,
    p2.product_category_name AS categoria_B,
    COUNT(*) AS frequencia
ORDER BY frequencia DESC
LIMIT 10;
```

Esta implementação demonstra a eficiência do Cypher na exploração de relações entre entidades, permitindo identificar padrões de compra conjunta de forma simples e menos verbosa do que numa abordagem relacional tradicional.

4. Síntese de Resultados

No âmbito do Momento 1 (**M1**), realizou-se a modelação e implementação de uma base de dados relacional com base no dataset Brazilian E-Commerce Public Dataset by Olist, tendo sido posteriormente executadas as questões analíticas definidas na secção 2.3.

A implementação do modelo relacional permitiu estruturar os dados de forma consistente e normalizada, garantindo a coerência das relações entre os dados, através de uma atribuição adequada de chaves primárias e estrangeiras. A separação entre as entidades Order e Order_item é particularmente interessante, permitindo representar corretamente a relação muitos-para-muitos entre encomendas e produtos, sem redundância desnecessária.

Neste enquadramento, a **pergunta 1**, a análise da receita por categoria evidenciou os segmentos de produtos que geram maior impacto na faturação global, permitindo identificar áreas estratégicas para o negócio.

Paralelamente, a identificação do produto "best seller" em cada estado revelou diferenças regionais no consumo (**Pergunta 2**).

A **pergunta 3** permitiu mapear as rotas comerciais mais frequentes entre vendedores e clientes, contribuindo para uma melhor compreensão da dinâmica logística da plataforma.

Relativamente ao ciclo de venda, a **pergunta 4** evidenciou que a maioria das encomendas se encontra no estado delivered, permitindo ainda quantificar o impacto financeiro associado às encomendas canceladas através do cálculo da faturação potencial perdida. A **pergunta 5** identificou ou 10 clientes que geram maior receita total, evidenciando a relevância de analisar a concentração de valor por utilizador.

Quanto à **questão 6**, dedicada à análise de produtos frequentemente comprados em conjunto, permitiu identificar padrões de Cross-selling dentro das encomendas, que revelaram informação analítica relevante para o negócio. Embora a frequência mais elevada ocorra entre produtos da mesma categoria (como no caso de 'beleza_saude'), destaca-se a presença de pares compostos por produtos de categorias totalmente diferentes no topo do ranking. Isto permite à Olist otimizar o seu motor de recomendações e criar campanhas de marketing muito mais direcionadas e eficazes.

O principal desafio técnico ocorreu na importação dos ficheiros CSV, devido à presença de valores vazios em colunas numéricas e temporais. Este problema foi resolvido com o comando LOAD DATA, utilizando validações para tornar valores vazios em NULL, com o apoio de ferramentas de inteligência artificial para otimização do processo.

Em relação ao **M2**, A transição do modelo relacional (MySQL) para o colunar (Apache Cassandra) exigiu uma adaptação estrita ao princípio de "Query-First Design", o que prioriza a otimização de leitura em detrimento da flexibilidade das consultas ad-hoc. A dificuldade mais sentida foi o facto de a Apache Cassandra não suportar operações de agregação como SUM nem junções (JOIN) de forma eficiente no momento da leitura. Por esse motivo, foi necessário realizar o pré-processamento dos dados no sistema relacional (MySQL). Além disso, na **Questão 1 e 2** houve um desafio técnicos comum: a importação de dados esbarrou na rigidez do Cassandra, que rejeitou a inserção de Primary Keys vazias (categorias nulas). Este obstáculo foi contornado com pré-processamento no MySQL, utilizando a instrução CASE WHEN para substituir valores inexistentes por 'Sem_Categoria'. Na implementação da **Questão 1,** para suprimir a incapacidade do Cassandra em executar agregações e ordenações no momento da leitura, a tabela foi projetada com Clustering Columns (WITH CLUSTERING ORDER BY), garantindo que os dados de receita fossem gravados no disco de forma já sequencial e descendente. Relativamente à **Questão 2**, focada na identificação do "Best Seller" por estado, a adoção da metodologia Query First exigiu que os dados fossem previamente filtrados no MySQL (recorrendo ao cálculo do valor máximo por região). A modelação no Cassandra utilizou uma Partition Key estática (id_fixo = 'Geral') e Clustering Columns ordenadas por quantidade, o que garantiu que o campeão de vendas de cada estado ficasse imediatamente disponível para leitura. Nesta fase, além do desafio previamente mencionado, é importante mencionar a resolução de erros de interrupção por limite de tempo (DriverTimeoutException) no DBeaver, corrigidos através do ajuste manual das propriedades de tolerância (request timeout) no driver da ligação.

Na **Questão 3** a maior dificuldade foi a definição correta das chaves (Partition Key e Clustering Columns). Inicialmente, não estava totalmente claro como estruturar a tabela de forma eficiente para suportar as consultas pretendidas. Esta dificuldade foi ultrapassada ao analisar o padrão de acesso necessário para a questão, concluindo-se que os dados deveriam ser organizados por ano e depois ordenados por estados de origem e destino. Relativamente à **Questão 4**, a principal dificuldade esteve no cálculo de métricas como a taxa de sucesso

e a faturação perdida, que exigiram o uso de expressões condicionais (SUM(IF(...))) e a prevenção de divisões por zero com NULLIF. Como a análise é feita por categoria e não em formato de ranking, não foi necessário definir Clustering Columns. Foi ainda aplicado um filtro para considerar apenas categorias com encomendas entregues ou canceladas. Quanto à **Questão 5**, a principal dificuldade nesta questão foi a necessidade da construção de um ranking global exigiu a definição de uma chave de partição fixa e ordenação por receita, o que implicou um cuidado extra no desenho da tabela. Já na **Questão 6**, focada em produtos comprados em conjunto, a limitação deveu-se à inexistência de Joins no Cassandra e à impossibilidade de definir atributos duplicados na mesma tabela, algo que o Self-Join do MySQL produzia visualmente. A solução consistiu em armazenar o resultado pré-calculado numa Column Family dedicada, distinguindo e renomeando explicitamente os campos de cada par na origem (ex: id_produto_a e id_produto_b). Esta adaptação gerou um modelo de dados mais rigoroso, isento de ambiguidades semânticas e pronto para extração imediata.

No Momento 3 (**M3**), ao comparar o modelo relacional e colunar com a implementação em Neo4j, observaram-se diferenças significativas na forma de representação e exploração dos dados. Enquanto o modelo relacional organiza a informação em tabelas normalizadas, recorrendo a chaves primárias e estrangeiras para estabelecer relações, e o Cassandra privilegia uma estrutura orientada às consultas com foco em desempenho e escalabilidade, o Neo4j adota um modelo orientado a grafos baseado em nós e relações. Esta abordagem permite representar de forma mais natural e direta as ligações entre entidades, tornando mais intuitiva a execução de consultas sobre relações complexas. No entanto, a transição para o Neo4j exigiu uma mudança de paradigma, substituindo o conceito de joins e estruturas tabulares por ligações no grafo e pela utilização da linguagem Cypher para modelar e consultar os dados.

Na **pergunta 1**, o armazenamento do preço e taxa de envio diretamente na relação [:CONTEM] eliminou tabelas intermédias e acelerou a pesquisa no grafo. Outra diferença notada foi a consulta simplificada pelo agrupamento implícito do Cypher (que dispensou o GROUP BY). Para garantir o rigor analítico num modelo schema-free, mesmo que não gerasse erros perante a ausência destes dados, aplicou-se um CASE WHEN para agregar produtos nulos sob a etiqueta 'Sem_Categoria'.

Na **pergunta 2**, a principal dificuldade foi a transição da lógica de subqueries do SQL para o mapeamento de percursos no grafo. Foi necessário garantir que o caminho entre as entidades (Customer, Order e Product) refletia fielmente o fluxo de vendas para que a contagem por estado fosse rigorosa. Houve também alguma dificuldade em isolar apenas o produto líder de cada região sem recorrer ao processamento externo. Esta limitação foi superada através da ordenação decrescente das quantidades combinada com a função collect()[0], que permitiu agrupar os dados e extrair apenas o "Best Seller" de cada estado de forma automática e eficiente na consulta.

Na **pergunta 3**, a principal dificuldade foi a definição correta da direção das relações no padrão de correspondência do grafo, uma vez que uma orientação incorreta entre nós (Customer, Order e Seller) podia levar à ausência de resultados ou a resultados incorretos. Foi também necessário compreender bem a estrutura do modelo de dados para garantir que o percurso no grafo refletia corretamente o fluxo entre entidades. Adicionalmente, houve alguma dificuldade na aplicação do filtro temporal sobre o atributo de data, de forma a garantir que a filtragem por ano fosse corretamente interpretada na consulta.

Na **pergunta 4**, a principal dificuldade centrou-se na correta aplicação do filtro temporal sobre o atributo de data e na posterior validação cruzada dos resultados com os restantes modelos. Inicialmente, a consulta falhava ao interpretar o campo order_purchase_timestamp como texto padrão; contudo, a limitação foi superada ao identificar que o Neo4j armazena esta propriedade como um tipo temporal nativo (DateTime com fuso horário), exigindo o isolamento explícito do componente .year na cláusula WHERE.

Adicionalmente, durante a auditoria dos resultados, o grupo enfrentou o desafio de justificar uma ligeira flutuação no volume de itens entregues face ao modelo relacional. Esta divergência foi desmistificada através da análise arquitetónica da ingestão: enquanto o MySQL barra duplicados através de restrições rígidas de Chave Primária, o ecossistema schema-free do Neo4j permitiu a criação de múltiplas relações [:CONTEM] paralelas para os mesmos nós, além de processar de forma distinta os registos situados nas franjas limítrofes de fuso horário. A resolução desta questão consolidou a capacidade do grupo em diagnosticar as nuances de consistência de dados entre paradigmas distintos.

Na **pergunta 5**, a principal dificuldade esteve relacionada com o cálculo correto da receita total por cliente, exigindo a navegação entre diferentes entidades do grafo através do percurso (Customer)-[:FAZ]->(Order)-[:CONTEM]->(Product). Além disso, foi necessário compreender que os valores de price e freight_value estavam armazenados diretamente na relação [:CONTEM].
Na **pergunta 6**, a principal dificuldade centrou-se na modelação correta da associação entre produtos comprados na mesma encomenda, uma vez que no Neo4j foi necessário compreender como percorrer o mesmo nó Order para relacionar múltiplos produtos através da relação [:CONTEM]. Ao contrário do modelo relacional, onde esta análise é normalmente feita com auto-joins sobre a tabela de itens da encomenda, a transição para um modelo orientado a grafos exigiu uma adaptação da lógica de raciocínio para percursos no grafo.
Outra dificuldade identificada foi evitar a geração de pares redundantes de produtos, como combinações invertidas do mesmo conjunto ou associações inválidas entre um produto e ele próprio. Foi necessário garantir que a estrutura da consulta não comprometesse a coerência dos resultados analíticos.
Adicionalmente, surgiu alguma dificuldade na validação da frequência das compras conjuntas, nomeadamente em assegurar que os produtos analisados pertenciam efetivamente à mesma encomenda e que os padrões identificados correspondiam a relações reais de compra entre categorias e produtos.

Para perceber qual o método mais eficiente comparamos os tempos de execução entre o MySQL e o Neo4j. O modelo colunar (Apache Cassandra) foi deliberadamente excluído, uma vez que a sua implementação exigiu o pré-processamento e agregação dos dados no MySQL, medir apenas o seu tempo de leitura de dados já processados não permitiria uma comparação arquitetónica justa.

Questão	Duração SQL	Duração Neo4j
Questão 1	0.316 (view+querie)	0.337
Questão 2	3.309	1.938
Questão 3	0.75	1.434
Questão 4	0.917	1.606
Questão 5	0.92	0.722
Questão 6	0.456 (view+querie)	0.569

Tabela 31 – Comparação da duração entre SQL e Neo4j

Como podemos observar na tabela 31, a avaliação dos tempos de execução entre o MySQL e o Neo4j permitiu concluir que a eficiência de cada modelo está estritamente dependente da natureza da consulta analítica:

- **Questões 1 e 6 (Limitação Metodológica):** Os tempos de execução destas consultas não são diretamente comparáveis. No modelo relacional, a utilização de Views reduziu a operação à leitura de uma estrutura previamente processada pelo sistema. Em contraste, o Neo4j realizou uma leitura integral a partir do zero, inviabilizando uma comparação arquitetónica justa.
- **Questões 2 e 5 (Vantagem do Modelo de Grafos):** O Neo4j demonstrou uma clara superioridade temporal. Estas questões exigem a exploração de percursos entre várias entidades (cliente, encomenda, produto) e a extração de hierarquias locais (como o melhor produto por estado). Nestes cenários, o modelo de grafos é altamente eficiente, enquanto o modelo relacional perde desempenho devido à sobrecarga de múltiplas subconsultas e cruzamento sucessivo de tabelas.
- **Questões 3 e 4 (Supremacia do Modelo Relacional):** O MySQL obteve o melhor desempenho em operações de agregação global e estatística de larga escala. O modelo de grafos revelou-se menos eficiente nestas questões porque a aplicação de filtros de datas e contagens condicionais exigiu a leitura exaustiva de todos os registos do sistema. Por sua vez, o modelo relacional possui algoritmos tradicionalmente otimizados para agrupar e processar volumes massivos de dados de forma mais rápida.

5. Avaliação do Grupo de Trabalho

A metodologia adotada pelo grupo para a atribuição das classificações individuais baseou-se nos princípios de equidade, responsabilidade partilhada e cumprimento rigoroso dos objetivos definidos.

A distribuição das tarefas foi organizada de forma que cada elemento assumisse a responsabilidade por componentes específicas do projeto, nomeadamente a Modelação, a Implementação, o Carregamento de Dados e a Análise de Resultados, o que garantiu que todas as áreas fossem desenvolvidas com a mesma profundidade técnica.

A avaliação individual considerou o cumprimento dos prazos estabelecidos, a qualidade técnica das contribuições, a colaboração entre os elementos e a participação ativa nas diferentes fases do trabalho.

Todos os membros entregaram as suas partes dentro do cronograma, respeitaram os requisitos de integridade e as normas definidas, mantiveram uma cooperação constante, especialmente na resolução de problemas críticos como os erros de importação, cuja solução foi desenvolvida em conjunto com o apoio da Inteligência Artificial.

Para além disso, todos participaram nas reuniões de coordenação e na revisão final do relatório.

Tendo em conta este desempenho uniforme e exemplar, o grupo decidiu, por unanimidade, atribuir a classificação de n a todos os seus elementos.

Estudante	Avaliação	Assinatura Digital
Ana Íris Machado Torres	*n*	Ana Íris Machado Torres — Assinado de forma digital por Ana Íris Machado Torres Dados: 2026.05.21 21:36:39 +01'00'
Daniela Gomes Paulino	*n*	Daniela Gomes Paulino — Assinado de forma digital por Daniela Gomes Paulino Dados: 2026.05.21 21:37:16 +01'00'
Maria Leonor Gomes Sampaio	*n*	Maria Leonor Gomes Sampaio — Assinado de forma digital por Maria Leonor Gomes Sampaio Dados: 2026.05.21 21:38:03 +01'00'
Maria Nair da Cunha e Silva Lacerda Ângelo	*n*	Maria Nair da Cunha e Silva Lacerda Ângelo — Assinado de forma digital por Maria Nair da Cunha e Silva Lacerda Ângelo Dados: 2026.05.21 21:38:36 +01'00'
Mariana Silva Gomes	*n*	Mariana Silva Gomes — Assinado de forma digital por Mariana Silva Gomes Dados: 2026.05.21 21:39:07 +01'00'

Tabela 32 – Avaliação do Grupo de Trabalho

6. Referências Bibliográficas

MAGIOLO,F., DABAGUE,L., SIONEK,A.(2022), *Brazilian E-Commerce Public Dataset by Olist, Kaggle.* Consultado em https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce?resource=download&select=olist_sellers_dataset.csv a 19 de fevereiro de 2026.

ANEXOS

ANEXO M1 – SQL

SQL

CREATE TABLES

```sql
USE bd2;

DROP TABLE IF EXISTS Order_item;
DROP TABLE IF EXISTS `Order`;
DROP TABLE IF EXISTS Product;
DROP TABLE IF EXISTS Seller;
DROP TABLE IF EXISTS Customer;

-- 1. Tabela de clientes (não dependem de ninguém)
CREATE TABLE Customer (
    customer_id VARCHAR(32) NOT NULL,
    customer_unique_id VARCHAR(32),
    customer_zip_code_prefix INT,
    customer_city VARCHAR(100),
    customer_state CHAR(2),
    PRIMARY KEY (customer_id)
);

-- 2. Tabela vendedores (não depende de ninguém)
CREATE TABLE Seller (
    seller_id VARCHAR(32) NOT NULL,
    seller_zip_code_prefix INT,
    seller_city VARCHAR(100),
    seller_state CHAR(2),
    PRIMARY KEY (seller_id)
);

-- 3. Tabela de produtos (não depende de ninguém)
CREATE TABLE Product (
    product_id VARCHAR(32) NOT NULL,
    product_category_name VARCHAR(100),
    product_name_length INT,
    product_description_length INT,
    product_photos_qty INT,
    product_weight_g INT,
    product_length_cm INT,
    product_height_cm INT,
    product_width_cm INT,
    PRIMARY KEY (product_id)
);

-- 4. Tabela de Encomendas (depende da Customer)
CREATE TABLE `Order` (
    order_id VARCHAR(32) NOT NULL,
    customer_id VARCHAR(32) NOT NULL,
    order_status VARCHAR(20),
    order_purchase_timestamp DATETIME,
    order_approved_at DATETIME,
    order_delivered_carrier_date DATETIME,
    order_delivered_customer_date DATETIME,
    order_estimated_delivery_date DATETIME,
    PRIMARY KEY (order_id),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

-- 5. Tabela de Itens (depende de Order, Product e Seller)
```

```sql
CREATE TABLE Order_item (
    order_id VARCHAR(32) NOT NULL,
    order_item_id INT NOT NULL,
    product_id VARCHAR(32) NOT NULL,
    seller_id VARCHAR(32) NOT NULL,
    shipping_limit_date DATETIME,
    price DECIMAL(10,2),
    freight_value DECIMAL(10,2),
    PRIMARY KEY (order_id, order_item_id),
    FOREIGN KEY (order_id) REFERENCES `Order`(order_id),
    FOREIGN KEY (product_id) REFERENCES Product(product_id),
    FOREIGN KEY (seller_id) REFERENCES Seller(seller_id)
);
```

INSERTS

```sql
USE bd2;

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

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

LOAD DATA INFILE '/var/lib/mysql/csv/olist_products_dataset.csv'
INTO TABLE Product
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(product_id, product_category_name, @vname_len, @vdesc_len, @vphotos_qty, @vweight, @vlength,
@vheight, @vwidth)
SET

product_name_length = IF(@vname_len = '', NULL, @vname_len),
product_description_length = IF(@vdesc_len = '', NULL, @vdesc_len),
product_photos_qty = IF(@vphotos_qty = '', NULL, @vphotos_qty),
product_weight_g = IF(@vweight = '', NULL, @vweight),
product_length_cm = IF(@vlength = '', NULL, @vlength),
product_height_cm = IF(@vheight = '', NULL, @vheight),
product_width_cm = IF(@vwidth = '', NULL, @vwidth);

LOAD DATA INFILE '/var/lib/mysql/csv/olist_orders_dataset.csv'
INTO TABLE `Order`
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(order_id, customer_id, order_status, @v_purchase, @v_approved, @v_carrier, @v_customer,
@v_estimated)
SET

order_purchase_timestamp = IF(@v_purchase = '', NULL, @v_purchase),
```

```
order_approved_at = IF(@v_approved = '', NULL, @v_approved),
order_delivered_carrier_date = IF(@v_carrier = '', NULL, @v_carrier),
order_delivered_customer_date = IF(@v_customer = '', NULL, @v_customer),
order_estimated_delivery_date = IF(@v_estimated = '', NULL, @v_estimated);

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

PERGUNTAS

1.

```
CREATE VIEW Receita_por_categoria AS
SELECT P.product_category_name,
       SUM(OI.price + OI.freight_value) AS receita_total
FROM Order_item OI
JOIN Product P ON OI.product_id = P.product_id
GROUP BY P.product_category_name;

SELECT *
FROM Receita_por_categoria
ORDER BY receita_total DESC
LIMIT 10;
```

2.

```
SELECT
    c.customer_state AS Estado,
    p.product_id AS ID_Produto,
    p.product_category_name AS Categoria,
    COUNT(oi.product_id) AS Quantidade_Vendida
FROM Customer c
JOIN `Order` o ON c.customer_id = o.customer_id
JOIN Order_item oi ON o.order_id = oi.order_id
JOIN Product p ON oi.product_id = p.product_id
GROUP BY c.customer_state, p.product_id, p.product_category_name
HAVING (c.customer_state, COUNT(oi.product_id)) IN (
    -- esta subquery encontra a quantidade máxima vendida em cada estado
    SELECT estado_sub, MAX(total_vendas)
    FROM (
        SELECT
            c2.customer_state AS estado_sub,
            COUNT(oi2.product_id) AS total_vendas
        FROM Customer c2
        JOIN `Order` o2 ON c2.customer_id = o2.customer_id
        JOIN Order_item oi2 ON o2.order_id = oi2.order_id
        GROUP BY c2.customer_state, oi2.product_id
    ) AS sub_tabela
    GROUP BY estado_sub
)
ORDER BY Quantidade_Vendida DESC;
```

3.

```
SELECT c.customer_state, s.seller_state,  COUNT(distinct o.order_id) AS total_fluxo
FROM Customer c
```

```sql
JOIN `Order` o ON c.customer_id=o.customer_id
JOIN Order_item oi ON o.order_id=oi.order_id
JOIN Seller s ON oi.seller_id=s.seller_id
WHERE o.order_purchase_timestamp BETWEEN '2016-01-01 00:00:00' AND '2016-12-31 23:59:59'
GROUP BY s.seller_state, c.customer_state
ORDER BY total_fluxo DESC;
```

4.

```sql
SELECT
    p.product_category_name AS Categoria,
    SUM(IF(o.order_status = 'delivered', 1, 0)) AS Total_Entregue,
    SUM(IF(o.order_status = 'canceled', 1, 0)) AS Total_Cancelado,
    -- Cálculo da Taxa de Sucesso: (Entregues / (Entregues + Cancelados))
    ROUND(
        SUM(IF(o.order_status = 'delivered', 1, 0)) /
            NULLIF(SUM(IF(o.order_status = 'delivered' OR o.order_status = 'canceled', 1, 0)),
0) * 100, 2) AS Taxa_Sucesso_Percent,
    -- Soma do preço + taxa de envio apenas para os cancelados (Faturação Perdida)
    SUM(IF(o.order_status = 'canceled', oi.price + oi.freight_value, 0)) AS Faturacao_Perdida
FROM
    Order_item oi
JOIN
    `Order` o ON oi.order_id = o.order_id
JOIN
    Product p ON oi.product_id = p.product_id
WHERE
    YEAR(o.order_purchase_timestamp) IN (2017, 2018)
GROUP BY
    p.product_category_name
HAVING
    Total_Entregue > 0 OR Total_Cancelado > 0
ORDER BY
    Faturacao_Perdida DESC;
```

5.

```sql
SELECT c.customer_unique_id,
    SUM(oi.price + oi.freight_value) AS receita_total
FROM `Order` o
JOIN customer c
    ON c.customer_id = o.customer_id
JOIN order_item oi
    ON oi.order_id = o.order_id
WHERE o.order_purchase_timestamp
      BETWEEN '2016-01-01 00:00:00'
          AND '2018-12-31 23:59:59'
  AND o.order_status = 'delivered'
GROUP BY c.customer_unique_id
ORDER BY receita_total DESC
LIMIT 10;
```

6.

```sql
CREATE OR REPLACE VIEW vw_pares_produtos_exatos AS
SELECT
    oi1.product_id AS id_produto_A,
    oi2.product_id AS id_produto_B,
    COUNT(*) AS vezes_comprados_juntos
FROM
    Order_item oi1
```

```sql
JOIN
    Order_item oi2 ON oi1.order_id = oi2.order_id
WHERE
    oi1.product_id < oi2.product_id
GROUP BY
    oi1.product_id, oi2.product_id
ORDER BY
    vezes_comprados_juntos DESC;

SELECT
    oi1.product_id,
    p1.product_category_name,
    oi2.product_id,
    p2.product_category_name,
    COUNT(*) AS frequencia
FROM Order_item oi1
JOIN Order_item oi2 ON oi1.order_id = oi2.order_id
JOIN Product p1 ON oi1.product_id = p1.product_id
JOIN Product p2 ON oi2.product_id = p2.product_id
WHERE oi1.product_id < oi2.product_id
GROUP BY oi1.product_id, oi2.product_id, p1.product_category_name, p2.product_category_name
ORDER BY frequencia DESC
LIMIT 10;
```

ANEXO M2 – Cassandra

CASSANDRA

PERGUNTA 1

EXPORTAÇÃO DE DADOS (MySQL):

```
SELECT
    'Geral' AS filtro,
    CASE
        WHEN p.product_category_name IS NULL OR p.product_category_name = '' THEN
'Sem_Categoria'
        ELSE p.product_category_name
    END AS categoria,
    SUM(oi.price + oi.freight_value) AS receita_total
FROM Order_item oi
JOIN Product p ON oi.product_id = p.product_id
GROUP BY p.product_category_name
INTO OUTFILE '/var/lib/mysql/csv/pergunta1.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

CRIAÇÃO DO KEYSPACE E TABELAS (Cassandra):

```
DROP KEYSPACE IF EXISTS bd2ks;

CREATE KEYSPACE bd2ks WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1};

USE bd2ks;

CREATE TABLE top_receita (
    id_fixo text,
    receita_total decimal,
    categoria text,
    PRIMARY KEY (id_fixo, receita_total, categoria)
) WITH CLUSTERING ORDER BY (receita_total DESC);
```

IMPORTAÇÃO DE DADOS (Cassandra):

```
USE bd2ks;
COPY top_receita(id_fixo, categoria, receita_total)
FROM '/var/lib/cassandra/pergunta1.csv'
WITH DELIMITER=';';
```

QUESTÃO ANALÍTICA (CQL):

```
USE bd2ks;

SELECT categoria, receita_total
FROM top_receita
WHERE id_fixo = 'Geral'
LIMIT 10;
```

PERGUNTA 2

EXPORTAÇÃO DE DADOS (MySQL):

```sql
SELECT
    'Geral' AS id_fixo,
    c.customer_state AS estado,
    oi.product_id AS id_produto,
    CASE
        WHEN p.product_category_name IS NULL OR p.product_category_name = '' THEN
'Sem_Categoria'
        ELSE p.product_category_name
    END AS categoria,
    COUNT(oi.product_id) AS quantidade_vendida
FROM Customer c
JOIN `Order` o ON c.customer_id = o.customer_id
JOIN Order_item oi ON o.order_id = oi.order_id
JOIN Product p ON oi.product_id = p.product_id
GROUP BY c.customer_state, oi.product_id, p.product_category_name
HAVING (c.customer_state, COUNT(oi.product_id)) IN (
    -- Subquery para encontrar o máximo vendido por estado
    SELECT estado_sub, MAX(total_vendas)
    FROM (
        SELECT c2.customer_state AS estado_sub, COUNT(oi2.product_id) AS total_vendas
        FROM Customer c2
        JOIN `Order` o2 ON c2.customer_id = o2.customer_id
        JOIN Order_item oi2 ON o2.order_id = oi2.order_id
        GROUP BY c2.customer_state, oi2.product_id
    ) AS sub_tabela
    GROUP BY estado_sub
)
ORDER BY quantidade_vendida DESC
INTO OUTFILE '/var/lib/mysql/csv/questao2.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

CRIAÇÃO DO KEYSPACE E TABELAS (Cassandra):

```cqlsh
DROP KEYSPACE IF EXISTS bd2ks;

CREATE KEYSPACE bd2ks WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1};

USE bd2ks;

CREATE TABLE best_seller_estado (
    id_fixo text,
    quantidade_vendida int,
    estado text,
    id_produto text,
    categoria text,
    PRIMARY KEY (id_fixo, quantidade_vendida, estado, id_produto)
) WITH CLUSTERING ORDER BY (quantidade_vendida DESC, estado ASC, id_produto ASC);
```

IMPORTAÇÃO DE DADOS (Cassandra):

```cqlsh
USE bd2ks;

COPY best_seller_estado (id_fixo, estado, id_produto, categoria, quantidade_vendida)
FROM '/var/lib/cassandra/questao2.csv'
WITH DELIMITER=';';
```

QUESTÃO ANALÍTICA (CQL):

```
USE bd2ks;

SELECT estado, id_produto, categoria, quantidade_vendida
FROM best_seller_estado
WHERE id_fixo = 'Geral'
LIMIT 6;
```

PERGUNTA 3

EXPORTAÇÃO DE DADOS (MySQL):

```
SELECT
    YEAR(o.order_purchase_timestamp) AS ano,
    c.customer_state,
    s.seller_state,
    COUNT(DISTINCT o.order_id) AS total_fluxo
FROM Customer c
JOIN `Order` o ON c.customer_id = o.customer_id
JOIN Order_item oi ON o.order_id = oi.order_id
JOIN Seller s ON oi.seller_id = s.seller_id
GROUP BY ano, s.seller_state, c.customer_state
INTO OUTFILE '/var/lib/mysql/csv/pergunta3.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

CRIAÇÃO DO KEYSPACE E TABELAS (Cassandra):

```
DROP KEYSPACE IF EXISTS p3;

CREATE KEYSPACE p3 WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1};

USE p3;

CREATE TABLE fluxo_vendas (
    ano text,
    total_fluxo int,
    seller_state text,
    customer_state text,
    PRIMARY KEY (ano, total_fluxo, seller_state, customer_state)
) WITH CLUSTERING ORDER BY (total_fluxo DESC);
```

IMPORTAÇÃO DE DADOS (Cassandra):

```
USE p3;

COPY fluxo_vendas (ano, customer_state, seller_state, total_fluxo)
FROM '/var/lib/cassandra/pergunta3.csv'
WITH DELIMITER=';';
```

QUESTÃO ANALÍTICA (CQL):

```
USE p3;

SELECT *
FROM fluxo_vendas
```

```
WHERE ano='2016'
LIMIT 10;
```

PERGUNTA 4

EXPORTAÇÃO DE DADOS (MySQL):

```
SELECT
    CASE
        WHEN p.product_category_name IS NULL OR p.product_category_name = '' THEN
'Sem_Categoria'
        ELSE p.product_category_name
    END AS categoria,
    SUM(IF(o.order_status = 'delivered', 1, 0)) AS total_entregue,
    SUM(IF(o.order_status = 'canceled', 1, 0)) AS total_cancelado,
    ROUND(
        SUM(IF(o.order_status = 'delivered', 1, 0)) /
        NULLIF(SUM(IF(o.order_status IN ('delivered', 'canceled'), 1, 0)), 0) * 100,
    2) AS taxa_sucesso,
    SUM(IF(o.order_status = 'canceled', oi.price + oi.freight_value, 0)) AS faturacao_perdida
FROM Product p
JOIN Order_item oi ON p.product_id = oi.product_id
JOIN `Order` o ON oi.order_id = o.order_id
WHERE YEAR(o.order_purchase_timestamp) IN (2017, 2018)
GROUP BY p.product_category_name
HAVING total_entregue > 0 OR total_cancelado > 0
INTO OUTFILE '/var/lib/mysql/csv/pergunta4.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

CRIAÇÃO DO KEYSPACE E TABELAS (Cassandra):

```
DROP KEYSPACE IF EXISTS bd2_vendas;

CREATE KEYSPACE bd2_vendas WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor':
1};

USE bd2_vendas;

CREATE TABLE bd2_vendas.sucesso_vendas (
    categoria text,
    total_entregue int,
    total_cancelado int,
    taxa_sucesso decimal,
    faturacao_perdida decimal,
    PRIMARY KEY (categoria)
```

IMPORTAÇÃO DE DADOS (Cassandra):

```
USE bd2_vendas;

COPY bd2_vendas.sucesso_vendas (categoria, total_entregue, total_cancelado, taxa_sucesso,
faturacao_perdida)
FROM '/var/lib/cassandra/pergunta4.csv'
WITH DELIMITER=';';
```

QUESTÃO ANALÍTICA (CQL):

```sql
USE bd2_vendas;

SELECT *
FROM bd2_vendas.sucesso_vendas;
```

PERGUNTA 5

EXPORTAÇÃO DE DADOS (MySQL):

```sql
SELECT
    'Geral' AS id_fixo,
    c.customer_unique_id,
    SUM(oi.price + oi.freight_value) AS total_receita
FROM `Order` o
JOIN customer c ON c.customer_id = o.customer_id
JOIN Order_item oi ON oi.order_id = o.order_id
WHERE o.order_status = 'delivered'
GROUP BY c.customer_unique_id
ORDER BY total_receita DESC
INTO OUTFILE '/var/lib/mysql/csv/pergunta5.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

CRIAÇÃO DO KEYSPACE E TABELAS (Cassandra):

```sql
DROP KEYSPACE IF EXISTS bd2_receita;

CREATE KEYSPACE bd2_receita
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};

USE bd2_receita;

CREATE TABLE top_clientes_receita (
    id_fixo text,
    total_receita decimal,
    customer_unique_id text,
    PRIMARY KEY (id_fixo, total_receita, customer_unique_id)
) WITH CLUSTERING ORDER BY (total_receita DESC);
```

IMPORTAÇÃO DE DADOS (Cassandra):

```sql
USE bd2_receita;

COPY top_clientes_receita (id_fixo, customer_unique_id, total_receita)
FROM '/var/lib/cassandra/pergunta5.csv'
WITH DELIMITER=';';
```

QUESTÃO ANALÍTICA (CQL):

```sql
USE bd2_receita;

SELECT
      customer_unique_id,
      total_receita,
FROM top_clientes_receita
WHERE id_fixo = 'Geral'
LIMIT 10;
```

PERGUNTA 6

EXPORTAÇÃO DE DADOS (MySQL):

```
SELECT
    'Geral' AS id_fixo,
    COUNT(*) AS frequencia,
    oi1.product_id AS id_produto_a,
    CASE
        WHEN p1.product_category_name IS NULL OR p1.product_category_name = '' THEN
'Sem_Categoria'
        ELSE p1.product_category_name
    END AS categoria_a,
    oi2.product_id AS id_produto_b,
    CASE
        WHEN p2.product_category_name IS NULL OR p2.product_category_name = '' THEN
'Sem_Categoria'
        ELSE p2.product_category_name
    END AS categoria_b
FROM Order_item oi1
JOIN Order_item oi2 ON oi1.order_id = oi2.order_id
JOIN Product p1 ON oi1.product_id = p1.product_id
JOIN Product p2 ON oi2.product_id = p2.product_id
WHERE oi1.product_id < oi2.product_id
GROUP BY oi1.product_id, oi2.product_id, p1.product_category_name, p2.product_category_name
ORDER BY frequencia DESC
INTO OUTFILE '/var/lib/mysql/csv/pergunta6.csv'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

CRIAÇÃO DO KEYSPACE E TABELAS (Cassandra):

```
DROP KEYSPACE IF EXISTS bd2ks;

CREATE KEYSPACE bd2ks WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1};

USE bd2ks;

CREATE TABLE pares_frequentes_juntos (
       id_fixo text,
       frequencia int,
    product_id_a text,
    categoria_a text,
    product_id_b text,
    categoria_b text,
    PRIMARY KEY (id_fixo, frequencia, product_id_a, product_id_b)
) WITH CLUSTERING ORDER BY (frequencia DESC);
```

IMPORTAÇÃO DE DADOS (Cassandra):

```
USE bd2ks;
COPY pares_frequentes_juntos(id_fixo, frequencia, product_id_a, categoria_a, product_id_b,
categoria_b)
FROM '/var/lib/cassandra/pergunta6.csv'
WITH DELIMITER=';';
```

QUESTÃO ANALÍTICA (CQL):

```
USE bd2ks;

SELECT product_id_a, categoria_a, product_id_b, categoria_b, frequencia
FROM pares_frequentes_juntos
WHERE id_fixo = 'Geral'
LIMIT 10;
```

ANEXO M3 – Neo4j

NEO4J

IMPORTAÇÃO E CRIAÇÃO DO GRAFO (Cypher):

```
CREATE CONSTRAINT FOR (c:Customer) REQUIRE c.customer_id IS UNIQUE;
CREATE CONSTRAINT FOR (o:Order) REQUIRE o.order_id IS UNIQUE;
CREATE CONSTRAINT FOR (s:Seller) REQUIRE s.seller_id IS UNIQUE;
CREATE CONSTRAINT FOR (p:Product) REQUIRE p.product_id IS UNIQUE;

LOAD CSV WITH HEADERS
FROM 'file:///olist_customers_dataset.csv' AS csvLineCustomer

CREATE (customer:Customer {
    customer_id: csvLineCustomer.customer_id,
    customer_unique_id: csvLineCustomer.customer_unique_id,
    customer_zip_code_prefix: csvLineCustomer.customer_zip_code_prefix,
    customer_city: csvLineCustomer.customer_city,
    customer_state: csvLineCustomer.customer_state
});


LOAD CSV WITH HEADERS
FROM 'file:///olist_orders_dataset.csv' AS csvLineOrder

CREATE (order:Order {
    order_id: csvLineOrder.order_id,
    order_status: csvLineOrder.order_status,
    order_purchase_timestamp: datetime(replace(csvLineOrder.order_purchase_timestamp,' ','T')),
    order_delivered_customer_date:
datetime(replace(csvLineOrder.order_delivered_customer_date,' ','T'))
})

WITH csvLineOrder, order

MATCH (customer:Customer {customer_id: csvLineOrder.customer_id})

CREATE (customer)-[:FAZ]->(order);


LOAD CSV WITH HEADERS
FROM 'file:///olist_sellers_dataset.csv' AS csvLineSeller

CREATE (seller:Seller {
    seller_id: csvLineSeller.seller_id,
    seller_zip_code_prefix: csvLineSeller.seller_zip_code_prefix,
    seller_city: csvLineSeller.seller_city,
    seller_state: csvLineSeller.seller_state
});


LOAD CSV WITH HEADERS FROM 'file:///olist_products_dataset.csv' AS csvLineProduct
CREATE(product:Product {
    product_id: csvLineProduct.product_id,
    product_category_name: csvLineProduct.product_category_name,
    product_name_length: toInteger(csvLineProduct.product_name_lenght),
    product_description_length: toInteger(csvLineProduct.product_description_lenght),
    product_photos_qty: toInteger(csvLineProduct.product_photos_qty),
    product_weight_g: toInteger(csvLineProduct.product_weight_g),
    product_length_cm: toInteger(csvLineProduct.product_length_cm),
    product_height_cm: toInteger(csvLineProduct.product_height_cm),
    product_width_cm: toInteger(csvLineProduct.product_width_cm)
});
```

```
LOAD CSV WITH HEADERS
FROM 'file:///olist_order_items_dataset.csv' AS csvLineOrderItem

MATCH (order:Order {order_id: csvLineOrderItem.order_id})

MATCH (product:Product {product_id: csvLineOrderItem.product_id})

MATCH (seller:Seller {seller_id: csvLineOrderItem.seller_id})

CREATE (seller)-[:EMITIU]->(order)

CREATE (order)-[:CONTEM {
    order_item_id: toInteger(csvLineOrderItem.order_item_id),
    shipping_limit_date:datetime(replace(csvLineOrderItem.shipping_limit_date,' ','T')),
    price: toFloat(csvLineOrderItem.price),
    freight_value: toFloat(csvLineOrderItem.freight_value)
}]->(product);
```

PERGUNTA 1

QUESTÃO ANALÍTICA (Cypher):

```
MATCH (o:Order)-[c:CONTEM]->(p:Product)
RETURN
    CASE
        WHEN p.product_category_name IS NULL OR p.product_category_name = '' THEN
'Sem_Categoria'
        ELSE p.product_category_name
    END AS categoria,
    sum(c.price + c.freight_value) AS receita_total ORDER BY receita_total DESC
LIMIT 10;
```

PERGUNTA 2

QUESTÃO ANALÍTICA (Cypher):

```
MATCH (c:Customer)-[:FAZ]->(o:Order)-[:CONTEM]->(p:Product)
WITH c.customer_state AS estado,
     p.product_id AS id_produto,
     p.product_category_name AS categoria,
     count(p) AS quantidade_vendida

ORDER BY quantidade_vendida DESC

WITH estado, collect({
    id: id_produto,
    categoria: categoria,
    quantidade: quantidade_vendida
})[0] AS best_seller

RETURN estado AS Estado,
       best_seller.id AS ID_Produto,
       best_seller.categoria AS Categoria,
       best_seller.quantidade AS Quantidade_Vendida
ORDER BY Quantidade_Vendida DESC;
```

PERGUNTA 3

QUESTÃO ANALÍTICA (Cypher):

```
MATCH (c:Customer)-[:FAZ]->(o:Order)<-[:EMITIU]-(s:Seller)
WHERE date(o.order_purchase_timestamp).year = 2016

RETURN c.customer_state AS customer_state, s.seller_state AS seller_state, count(DISTINCT
o.order_id) AS total_fluxo

ORDER BY total_fluxo DESC;
```

PERGUNTA 4

QUESTÃO ANALÍTICA (Cypher):

```
MATCH (o:Order)-[c:CONTEM]->(p:Product)
WHERE o.order_status IN ['delivered', 'canceled']

  AND o.order_purchase_timestamp.year IN [2017, 2018]

WITH
  CASE
    WHEN p.product_category_name IS NULL OR p.product_category_name = '' THEN 'Sem Categoria'
    ELSE p.product_category_name
  END AS categoria,
  o.order_status AS status,
  c.price + c.freight_value AS valor_item

WITH categoria,
     sum(CASE WHEN status = 'delivered' THEN 1 ELSE 0 END) AS total_entregue,
     sum(CASE WHEN status = 'canceled' THEN 1 ELSE 0 END) AS total_cancelado,
     sum(CASE WHEN status = 'canceled' THEN valor_item ELSE 0 END) AS faturacao_perdida

RETURN categoria,
       total_entregue,
       total_cancelado,
       CASE
         WHEN (total_entregue + total_cancelado) = 0 THEN 0.0
         ELSE (toFloat(total_entregue) / (total_entregue + total_cancelado)) * 100
       END AS taxa_sucesso,
       faturacao_perdida

ORDER BY faturacao_perdida DESC;
```

PERGUNTA 5

QUESTÃO ANALÍTICA (Cypher):

```
MATCH (c:Customer)-[:FAZ]->(o:Order)-[ct:CONTEM]->(p:Product)

WITH
    c.customer_id AS cliente,
    c.customer_city AS cidade,
    c.customer_state AS estado,
```

```
    sum(ct.price + ct.freight_value) AS receita_total

RETURN
    cliente,
    cidade,
    estado,
    receita_total

ORDER BY receita_total DESC
LIMIT 10;
```

PERGUNTA 6

QUESTÃO ANALÍTICA (Cypher):

```
MATCH (o:Order)-[:CONTEM]->(p1:Product), (o)-[:CONTEM]->(p2:Product)

WHERE p1.product_id < p2.product_id

RETURN
    p1.product_id AS produto_A,
    p1.product_category_name AS categoria_A,
    p2.product_id AS produto_B,
    p2.product_category_name AS categoria_B,
    COUNT(*) AS frequencia

ORDER BY frequencia DESC
LIMIT 10;
```