Projeto de Disciplina: Banco de Dados Relacional e Não-Relacional

Aluno: Marcelo Carvalho Vieira (marcelo.vieira@academico.unirv.edu.br)

Este documento detalha o "Projeto de Disciplina: Banco de Dados Relacional e Não-Relacional", focado na modelagem e implementação de soluções de banco de dados para diferentes necessidades de negócio. O projeto explora cinco tipos de bancos de dados, cada um com sua justificativa, dados modelados, responsabilidades e scripts de implementação e consulta.

Etapas do Projeto:

Etapa 1: Banco de Dados Relacional (PostgreSQL)

- **Responsabilidade:** Manter a consistência transacional (ACID) para dados críticos.
- **Dados Modelados:** Clientes, Pedidos, Itens de Pedido e Transações Financeiras.
- **Justificativa:** Integridade de dados fundamental.
- **Ferramentas:** PostgreSQL, pgAdmin4.
- **Scripts:** Criação de tabelas, inserção de dados de exemplo e consultas básicas.

Etapa 2: Banco de Dados de Documento (MongoDB)

- **Responsabilidade:** Armazenar dados com estrutura flexível ou semiestruturada.
- **Dados Modelados:** Catálogo de Produtos e Perfis estendidos de usuários.
- **Justificativa:** Flexibilidade do esquema para evolução de dados.
- **Ferramentas:** Docker, PowerShell, Navicat Premium.
- **Scripts:** Inicialização no Docker, criação de DB e inserção de coleções JSON, consultas `find().pretty()`.

Etapa 3: Banco de Dados Chave-Valor (Redis)

- **Responsabilidade:** Gerenciar dados voláteis com latência mínima.
- **Dados Modelados:** Sessões de usuário, carrinhos de compra e cache de dados frequentemente acessados.
- **Justificativa:** Desempenho para operações em tempo real.
- **Ferramentas:** Docker, PowerShell, Navicat Premium.
- **Scripts:** Inicialização no Docker, gerenciamento de sessões (`SET`, `GET`, `TTL`), carrinho de compras (`HSET`, `HGETALL`, `HINCRBY`, `HDEL`) e cache (`SET`, `GET`, `EX`), e contador de visualizações (`INCR`).

Etapa 4: Banco de Dados Colunar (ClickHouse)

- **Responsabilidade:** Ingestão e processamento de grandes volumes de dados para análises (OLAP).

- **Dados Modelados:** Logs de eventos da aplicação (visualizações, cliques, termos de busca).
- **Justificativa:** Otimizado para agregações e varreduras em larga escala.
- **Ferramentas:** Docker, PowerShell, interface web do ClickHouse.
- **Scripts:** Inicialização no Docker, criação de DB e tabela `event_logs` com `MergeTree`, inserção de dados de exemplo e consultas analíticas.

Etapa 5: Banco de Dados de Grafo (Neo4j)

- **Responsabilidade:** Mapear e consultar relacionamentos complexos, incluindo sistema de recomendação.
- **Dados Modelados:** Grafo de recomendações (Nós: Cliente, Produto, Marca, Categoria; Arestas: COMPROU, AVALIOU, VISUALIZOU).
- **Justificativa:** Eficiência para consultas baseadas em relacionamentos complexos.
- **Ferramentas:** Neo4j Desktop.
- **Scripts:** Criação de nós (Cliente, Marca, Categoria, Produto) e relacionamentos entre eles, além de consultas para verificar o grafo e identificar recomendações.

—----------

Etapa 1: Banco de Dados Relacional (PostgreSQL)

- **Responsabilidade:** Manter a consistência transacional (ACID) para os dados críticos do negócio.
- **Dados a serem modelados:** Clientes (dados cadastrais essenciais), Pedidos, Itens de Pedido e Transações Financeiras.
- **Justificativa:** A estrutura rígida e as garantias transacionais do modelo relacional são adequadas para dados onde a integridade é um requisito fundamental.

Para execução da etapa do Banco de Dados Relacional (PostgreSQL), foi utilizado da criação do banco de dados, tabelas e seus atributos, consultas a IDE pgAdmin4 para gerenciamento da parte gráfica do banco de dados PostgreSQL.

Não houve nenhuma dificuldade para execução dessa etapa, já que toda a parte de instalação e configuração do PostgreSQL e do pgAdmin4 é rápida e fácil de ser feita, e após o processo de instalação foi possível executar os scripts para criação do Schema dentro do Banco de dados.

Queries

1º Criação do Banco de Dados (Tabelas e atributos):

Este script cria as tabelas necessárias com seus respectivos relacionamentos (chaves primárias e estrangeiras), garantindo a integridade dos dados.

```sql
-- Tabela de Clientes, armazena os dados cadastrais essenciais dos usuários.
CREATE TABLE clientes (
    cliente_id SERIAL PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    cpf VARCHAR(14) NOT NULL UNIQUE,
    data_cadastro TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Tabela de Produtos.
CREATE TABLE produtos (
    produto_id SERIAL PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    sku VARCHAR(100) NOT NULL UNIQUE,
    preco_unitario NUMERIC(10, 2) NOT NULL,
    estoque INT NOT NULL CHECK (estoque >= 0),
    categoria VARCHAR(100)
);

-- Tabela de Pedidos, armazena o cabeçalho de cada pedido realizado.
CREATE TABLE pedidos (
    pedido_id SERIAL PRIMARY KEY,
    cliente_id INT NOT NULL,
    data_pedido TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    status VARCHAR(50) NOT NULL,
    valor_total NUMERIC(10, 2) NOT NULL,
    CONSTRAINT fk_cliente
        FOREIGN KEY(cliente_id)
        REFERENCES clientes(cliente_id)
);

-- Tabela de Itens de Pedido, que detalha os produtos contidos em cada pedido.
CREATE TABLE itens_pedido (
    item_id SERIAL PRIMARY KEY,
    pedido_id INT NOT NULL,
    produto_id INT NOT NULL,
    quantidade INT NOT NULL,
    preco_unitario NUMERIC(10, 2) NOT NULL, -- Preço no momento da compra
    CONSTRAINT fk_pedido
        FOREIGN KEY(pedido_id)
        REFERENCES pedidos(pedido_id),
    CONSTRAINT fk_produto
        FOREIGN KEY(produto_id)
        REFERENCES produtos(produto_id)
);
```

```
-- Tabela de Transações Financeiras, registra os detalhes de pagamento associados a cada pedido.
CREATE TABLE transacoes_financeiras (
    transacao_id SERIAL PRIMARY KEY,
    pedido_id INT NOT NULL,
    metodo_pagamento VARCHAR(50),
    valor NUMERIC(10, 2) NOT NULL,
    status_pagamento VARCHAR(50) NOT NULL,
    data_transacao TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    CONSTRAINT fk_pedido_transacao
        FOREIGN KEY(pedido_id)
        REFERENCES pedidos(pedido_id)
);
```

2º Inserção de dados de Exemplo nas Tabelas:

Este script popula as tabelas com dados de exemplo para simular um ambiente produtivo.

```
-- Inserindo Clientes
INSERT INTO clientes (nome, email, cpf) VALUES
('Marcelo Carvalho', 'marcelo.carvalho@gmail.com', '111.222.333-44'),
('Augusto Carrara', 'augusto_carrara@hotmail.com', '222.333.444-55'),
('Gerson Flamengo', 'gerson_flamengo@yahoo.com', '333.444.555-66');

-- Inserindo Produtos
INSERT INTO produtos (nome, sku, preco_unitario, estoque, categoria) VALUES
('Notebook Dell', '00001', 7500.00, 50, 'Eletrônicos'),
('Iphone 14', '00002', 3250.50, 120, 'Eletrônicos'),
('Apple Pods', '00003', 499.90, 300, 'Acessórios'),
('Cadeira DT3', '00004', 1200.00, 80, 'Móveis');

-- Simulação de um Pedido (Cliente 1 compra 1 Notebook e 2 Fones)
-- O valor total do pedido é: (1 * 7500.00) + (2 * 499.90) = 7500 + 999.80 = 8499.80
INSERT INTO pedidos (cliente_id, status, valor_total) VALUES (1, 'Processando', 8499.80)
INSERT INTO itens_pedido (pedido_id, produto_id, quantidade, preco_unitario) VALUES
(1, 1, 1, 7500.00),
(1, 3, 2, 499.90);
INSERT INTO transacoes_financeiras (pedido_id, metodo_pagamento, valor, status_pagamento) VALUES
(1, 'Cartão de Crédito', 8499.80, 'Aprovado');

-- Simulação de um segundo Pedido (Cliente 2 compra 1 Iphone)
-- O valor total do pedido é: 1 * 3250.50 = 3250.50
INSERT INTO pedidos (cliente_id, status, valor_total) VALUES (2, 'Enviado', 3250.50)
INSERT INTO itens_pedido (pedido_id, produto_id, quantidade, preco_unitario) VALUES
(2, 2, 1, 3250.50);
```

```
INSERT INTO transacoes_financeiras (pedido_id, metodo_pagamento, valor,
status_pagamento) VALUES
(2, 'PIX', 3250.50, 'Aprovado');

-- Simulação de um terceiro Pedido (Cliente 1 compra 1 Cadeira)
-- O valor total do pedido é: 1 * 1200.00 = 1200.00
INSERT INTO pedidos (cliente_id, status, valor_total) VALUES (1, 'Entregue', 1200.00)
INSERT INTO itens_pedido (pedido_id, produto_id, quantidade, preco_unitario) VALUES
(3, 4, 1, 1200.00);
INSERT INTO transacoes_financeiras (pedido_id, metodo_pagamento, valor,
status_pagamento) VALUES
(3, 'Boleto', 1200.00, 'Pendente');
```

3º Consultas:

Para confirmar que tudo foi criado e populado corretamente, foi executado as seguintes consultas SELECT nas tabelas:

```
SELECT * FROM clientes;
SELECT * FROM produtos;
SELECT * FROM pedidos;
SELECT * FROM itens_pedido;
SELECT * FROM transacoes_financeiras;
```

Etapa 2: Banco de Dados de Documento (MongoDB)

• **Responsabilidade:** Armazenar dados com estrutura flexível ou semiestruturada.

- **Dados a serem modelados:** Catálogo de Produtos, onde cada item pode possuir um conjunto distinto de atributos. Perfis estendidos de usuários (preferências, dados demográficos complementares).
- **Justificativa:** A flexibilidade do esquema (schema-on-read) facilita a evolução do catálogo e dos perfis de usuário sem a necessidade de migrações complexas na estrutura de dados.

Para realização da Etapa 2: Banco de Dados de Documento (MongoDB), foi necessário a instalação e configuração do **Docker** na máquina, onde também não houve dificuldades, o próprio site do Docker (https://www.docker.com) tem um passo-a-passo de como instalar e configurar a aplicação no sistema para rodar a imagem e ambiente de execução do MongoDB. Para execução dos Scripts, foi utilizado o PowerShell integrado do Windows e para visualização gráfica foi utilizado a IDE **Navicat Premium trial.**

Scripts:
1º Criação e Execução de imagem MongoDB no Docker (PowerShell):

Inicialização do MongoDB no Docker
docker run --name mongo-datastore -p 27017:27017 -d mongo

Acessar o Shell do MongoDB (mongosh)
docker exec -it mongo-datastore mongosh

2º Criação do Banco de dados e Inserção de dados em coleções JSON de Produtos e Perfis de usuários:

Gerar Banco de dados
use datastore

Inserção de dados em coleções JSON
Produtos:

```
db.produtos.insertMany([
  {
    sku: "00001",
    nome: "Notebook Dell",
    marca: "Dell",
    categoria_principal: "Eletrônicos",
    tags: ["gamer", "notebook", "alta performance"],
    preco: 7500.00,
    em_estoque: true,
    especificacoes_tecnicas: {
      processador: "Intel Core i9",
      memoria_ram: "32GB DDR5",
      armazenamento: "1TB SSD NVMe",
      placa_de_video: "NVIDIA RTX 4080"
    },
    avaliacoes: [
      {
        usuario_email: "marcelo.carvalho@gmail.com",
        nota: 5,
        comentario: "Máquina Top",
        data: new Date("2025-06-15T10:00:00Z")
      }
    ]
  },
  {
    sku: "00002",
    nome: "Iphone 14",
    marca: "Apple",
    categoria_principal: "Eletrônicos",
```

```
    tags: ["celular", "ios", "camera"],
    preco: 3250.50,
    em_estoque: true,
    especificacoes_tecnicas: {
      tela: "6.7 polegadas OLED",
      camera_traseira: "108 MP",
      bateria: "5000 mAh",
      armazenamento: "256GB"
    },
    avaliacoes: []
  },
  {
    sku: "00003",
    nome: "Apple Pods",
    marca: "Apple",
    categoria_principal: "Acessórios",
    tags: ["audio", "sem fio", "bluetooth"],
    preco: 499.90,
    em_estoque: true,
    especificacoes_tecnicas: {
      tipo: "Intraauricular",
      cancelamento_ruido: true,
      conectividade: "Bluetooth"
    },
    avaliacoes: [
      {
        usuario_email: "augusto_carrara@hotmail.com",
        nota: 1,
        comentario: "Muito ruim, escuto nada",
        data: new Date("2025-07-01T14:30:00Z")
      },
      {
        usuario_email: "gerson_flamengo@yahoo.com",
        nota: 5,
        comentario: "Excelente qualidade de som!",
        data: new Date("2025-07-02T09:00:00Z")
      }
    ]
  }
]);
```

Perfis de Usuários:

```
db.usuarios_extendidos.insertMany([
  {
    email: "augusto_carrara@hotmail.com",
    preferencias: {
      categorias: ["Acessórios", "Móveis", "Carros", "Taxi", "Uber"],
```

```
      marcas: ["Android"]
    },
    dados_demograficos: {
      cidade: "São Paulo",
      estado: "SP",
      faixa_etaria: "30-40"
    },
    lista_desejos: ["00002"]
  },
  {
    email: "marcelo.carvalho@gmail.com",
    preferencias: {
      categorias: ["Eletrônicos"],
      marcas: ["Dell"]
    },
    dados_demograficos: {
      cidade: "Goias",
      estado: "GO",
      faixa_etaria: "20-30"
    },
    lista_desejos: []
  }
]);
```

3º Consultas:

```
// Para ver todos os produtos em um formato legível
db.produtos.find().pretty()
// Para ver os perfis estendidos dos usuários
db.usuarios_extendidos.find().pretty()
```

Etapa 3: Banco de Dados Chave-Valor (Redis)

- **Responsabilidade:** Gerenciar dados voláteis que requerem acesso com latência mínima.
- **Dados a serem modelados:** Sessões de usuário, carrinhos de compra e cache de dados frequentemente acessados.
- **Justificativa:** Acesso a dados em memória proporciona o desempenho necessário para operações em tempo real que impactam diretamente a experiência do usuário.

Para realização da Etapa 3: Banco de Dados Chave-Valor (Redis), basta seguir os mesmos passos realizados da Etapa 2, o ambiente Redis será criado e executado também pelo **Docker**, seus scripts serão executados pelo **PowerShell** e sua visualização gráfica também ocorre pela IDE **Navicat Premium Trial.**

Scripts:
1º Criação e Execução de imagem Redis no Docker (PowerShell):

Inicialização do Redis no Docker
docker run --name redis-datastore -p 6379:6379 -d redis

Abrir o Redis
docker exec -it redis-container redis-cli

2º Gerenciamento de Sessão de usuários:

```
-- Simula o login dos usuários, criando uma sessão que expira em 30 minutos (1800
segundos).
-- Os tokens seriam gerados pela aplicação no momento do login.
SET session:a00001 "augusto_carrara@hotmail.com" EX 1800
SET session:a00002 "marcelo_carvalho@gmail.com" EX 1800
SET session:a00003 "gerson_flamengo@yahoo.com" EX 1800

# Para verificar qual usuário está associado a um token de sessão:
GET session:a00001
GET session:a00002
GET session:a00003

# Para verificar o tempo de vida restante da chave (em segundos):
TTL session:a00001
TTL session:a00002
TTL session:a00003
```

3º Gerenciamento de Carrinho de Compras:

```
-- O cliente com ID 2 adiciona 1 Iphone no seu carrinho.
HSET carrinho:user:2 00002 1

-- O mesmo cliente adiciona 2 fones de ouvido ao carrinho.
HSET carrinho:user:2 00003 2

-- Para visualizar todos os itens e quantidades no carrinho do cliente 2:
HGETALL carrinho:user:2

-- Marcelo desiste de uma unidade do fone. Diminuímos a quantidade.
HINCRBY carrinho:user:2 00003 -1

-- Para remover um item completamente do carrinho:
HDEL carrinho:user:2 00002
```

-- Verificando o estado final do carrinho:
HGETALL carrinho:user:2

4º Cache de Dados frequentemente acessados:

-- Uma aplicação buscaria este produto no MongoDB e o armazenaria no Redis por 5 minutos (300s) para acessos futuros.
SET product:cache:00001 '{"nome": "Notebook Dell", "preco": 7500.00, "categoria": "Eletrônicos"}' EX 300

-- Em uma requisição futura, a aplicação primeiro verificaria o cache:
GET product:cache:00001

5º Contador de Visualizações de Página de Produto:

-- Cada vez que um usuário visita a página do Notebook Dell, este comando é executado.
INCR views:product:00001
INCR views:product:00001
INCR views:product:00001

-- Para obter o número total de visualizações:
GET views:product:00001

-- Outro produto também é visualizado:
INCR views:product:00002

Etapa 4: Banco de Dados Colunar (ClickHouse)

- **Responsabilidade:** Ingestão e processamento de grandes volumes de dados para consul tas analíticas (OLAP).
- **Dados a serem modelados:** Logs de eventos da aplicação, como visualizações de produtos, cliques, termos de busca e outras interações do usuário.
- **Justificativa:** A arquitetura colunar é otimizada para agregações e varreduras em larga escala, sendo ideal para a geração de relatórios e dashboards analíticos.

Para realização da Etapa 4: Banco de Dados Colunar (ClickHouse), continuaremos com o processo de criação de imagem e ambiente pelo Docker e comando via PowerShell, porém para execução dos scripts relacionados ao Banco de dados diretamente, será realizado pela própria interface gráfica disponibilizada pelo clickhouse, onde em configuração padrão estará localizada na url: http://localhost:8123 (porta configurada no Docker).

Scripts:
1º Criação e Execução de imagem ClickHouse no Docker (PowerShell):

```
docker run -d --name clickhouse-server -p 8123:8123 -p 9000:9000 --ulimit
nofile=262144:262144 -e CLICKHOUSE_USER=default -e
CLICKHOUSE_PASSWORD="root" clickhouse/clickhouse-server
```

2º Criação do Banco de dados e da Tabela de Eventos:

```
-- Criando o banco de dados para nossas análises
CREATE DATABASE datastore

-- Criando a tabela para armazenar todos os logs de eventos utilizando a engine do
ClickHouse MergeTree (A mais utilizada quando o assunto é alta ingestão de dados sobre
grande demanda)
CREATE TABLE datastore.event_logs
(
    -- Momento em que o evento ocorreu
    event_time DateTime,

    -- Tipo do evento
    event_type String,

    -- Identificador do usuário (ID)
    user_email String,

    -- SKU do produto relacionado ao evento
    product_sku Nullable(String),

    -- Termo pesquisado pelo usuário
    search_query Nullable(String),

    -- Origem do tráfego (para análise de campanhas de marketing)
    traffic Nullable(String),

    -- Identificador da sessão do usuário
    session_id String
)
ENGINE = MergeTree()
PARTITION BY toYYYYMM(event_time)
ORDER BY (event_type, event_time);
```

3º Inserção de Dados de Exemplo:

```
-- Inserindo um fluxo de eventos que simula uma jornada de compra
INSERT INTO datastore.event_logs
(event_time, event_type, user_email, product_sku, search_query, traffic, session_id)
```

```
VALUES
-- Jornada do Marcelo Carvalho
    (now(), 'search', 'marcelo_carvalho@gmail.com', NULL, 'notebook gamer', 'google',
'abc1'),
    (now() + INTERVAL 2 SECOND, 'view_product', 'marcelo_carvalho@gmail.com', '00001',
NULL, 'google', 'abc1'),
    (now() + INTERVAL 5 SECOND, 'add_to_cart', 'marcelo_carvalho@gmail.com', '00001',
NULL, 'google', 'abc1'),
    (now() + INTERVAL 10 SECOND, 'view_product', 'marcelo_carvalho@gmail.com', '00003',
NULL, 'google', 'abc1'),
    (now() + INTERVAL 15 SECOND, 'purchase', 'marcelo_carvalho@gmail.com', NULL, NULL,
'google', 'abc1'),

-- Jornada do Augusto Carrara
    (now() - INTERVAL 1 DAY, 'search', 'augusto_carrara@hotmail.com', NULL, 'iphone',
'facebook', 'cde2'),
    (now() - INTERVAL 1 DAY + INTERVAL 3 SECOND, 'view_product',
'augusto_carrara@hotmail.com', '00002', NULL, 'facebook', 'cde2'),
    (now() - INTERVAL 1 DAY + INTERVAL 8 SECOND, 'add_to_cart',
'augusto_carrara@hotmail.com', '00002', NULL, 'facebook', 'cde2'),
    (now() - INTERVAL 1 DAY + INTERVAL 20 SECOND, 'purchase',
'augusto_carrara@hotmail.com', NULL, NULL, 'facebook', 'cde2'),

    -- Outros eventos de visualização para popular os dados
    (now() - INTERVAL 2 DAY, 'view_product', 'gerson_flamengo@yahoo.com', '00001', NULL,
'direct', 'efg3'),
    (now() - INTERVAL 2 DAY, 'view_product', 'marcelo.carvalho@gmail.com', '00003', NULL,
'direct', 'hij4'),
    (now(), 'search', 'gerson_flamengo@yahoo.com', NULL, 'cadeira dt3', 'google', 'klm5');
```

4º Consultas:

```
SELECT * FROM datastore.event_logs;
SELECT event_type, count() AS total_de_eventos
        FROM datastore.event_logs
        GROUP BY event_type
        ORDER BY total_de_eventos DESC;
```

Etapa 5: Banco de Dados de Grafo (Neo4j)

- **Responsabilidade:** Mapear e consultar relacionamentos complexos, incluindo a implementação de um sistema de recomendação.
- **Dados a serem modelados:** Grafo de recomendações, contendo nós como Cliente, Produto, Marca, e Categoria, e arestas representando interações como COMPROU, AVALIOU e VISUALIZOU. A implementação deve habilitar a

execução de algoritmos de filtragem colaborativa.

- **Justificativa:** A estrutura de grafo é eficiente para executar consultas baseadas em relacionamentos complexos, essenciais para sistemas de recomendação

Para realização da Etapa 5: Banco de Dados de Grafo (Neo4j), utilizaremos agora da própria interface disponibilizada pela aplicação Desktop, onde toda a parte de criação da instância e do próprio Banco de Dados é feita de forma rápida e intuitiva, e para execução dos Scripts é utilizado da linguagem CYPHER para criação do Nós e Relacionamentos.

Scripts:
1º Criação dos Nós (Cliente, Marca, Categoria, Produto):

```
CREATE  (c1:Cliente {nome: 'Marcelo Carvalho', email: 'marcelo_carvalho@gmail.com'});
CREATE  (c2:Cliente {nome: 'Augusto Carrara', email: 'augusto_carrara@hotmail.com'});
CREATE  (c3:Cliente {nome: 'Gerson Flamengo', email: 'gerson_flamengo@yahoo.com'});

CREATE (m1:Marca {nome: 'Dell'});
CREATE (m2:Marca {nome: 'Apple'});

CREATE (cat1:Categoria {nome: 'Eletrônicos'});
CREATE (cat2:Categoria {nome: 'Acessórios'});
CREATE (cat3:Categoria {nome: 'Móveis'});

CREATE (p1:Produto {nome: 'Notebook Dell', sku: '00001', preco: 7500.00});
CREATE (p2:Produto {nome: 'Iphone 14', sku: '00002', preco: 3250.50});
CREATE (p3:Produto {nome: 'Apple Pods', sku: '00003', preco: 499.90});
CREATE (p4:Produto {nome: 'Cadeira DT3', sku: '00004', preco: 1200.00});
```

2º Criação dos Relacionamentos (Representação das Interações):

```
-- Relacionar Produtos com suas Marcas e Categorias
MATCH (p:Produto {sku: '00001'}), (m:Marca {nome: 'Dell'}), (cat:Categoria {nome:
'Eletrônicos'}) MERGE (p)-[:PERTENCE_A]->(m) MERGE (p)-[:FAZ_PARTE_DE]->(cat);

MATCH (p:Produto {sku: '00002'}), (m:Marca {nome: 'Apple'}), (cat:Categoria {nome:
'Eletrônicos'}) MERGE (p)-[:PERTENCE_A]->(m) MERGE (p)-[:FAZ_PARTE_DE]->(cat);

MATCH (p:Produto {sku: '00003'}), (m:Marca {nome: 'Apple'}), (cat:Categoria {nome:
'Acessórios'}) MERGE (p)-[:PERTENCE_A]->(m) MERGE (p)-[:FAZ_PARTE_DE]->(cat);

MATCH (p:Produto {sku: '00004'}), (cat:Categoria {nome: 'Móveis'}) MERGE
(p)-[:FAZ_PARTE_DE]->(cat);
```

```
-- Criar relacionamentos de interações dos usuários (simulando os logs do ClickHouse e pedidos do PostgreSQL)
-- Interações de Marcelo Carvalho
MATCH (c:Cliente {email: 'marcelo_carvalho@gmail.com'}), (p:Produto {sku: '00001'})
CREATE (c)-[:VISUALIZOU {data: datetime('2025-07-03T13:30:00.000-03:00')}]->(p);

MATCH (c:Cliente {email: 'marcelo_carvalho@gmail.com'}), (p:Produto {sku: '00003'})
CREATE (c)-[:VISUALIZOU {data: datetime('2025-07-03T13:31:00.000-03:00')}]->(p);

-- Marcelo comprou o Notebook e o Fone no mesmo pedido
MATCH (c:Cliente {email: 'marcelo_carvalho@gmail.com'}), (p1:Produto {sku: '00001'}),
(p2:Produto {sku: '00003'})
CREATE (c)-[:COMPROU {pedido_id: 1, data:
datetime('2025-07-03T13:35:00.000-03:00')}]->(p1)
CREATE (c)-[:COMPROU {pedido_id: 1, data:
datetime('2025-07-03T13:35:00.000-03:00')}]->(p2);

-- Marcelo também avaliou o Fone de Ouvido
MATCH (c:Cliente {email: 'marcelo_carvalho@gmail.com'}), (p:Produto {sku: '00003'})
CREATE (c)-[:AVALIOU {nota: 4, comentario: 'Bom Som!.', data:
datetime('2025-07-04T10:00:00.000-03:00')}]->(p);

-- Interações de Augusto Carrara
MATCH (c:Cliente {email: 'augusto_carrara@hotmail.com'}), (p:Produto {sku: '00001'})
CREATE (c)-[:AVALIOU {nota: 5, comentario: 'Máquina incrível!', data:
datetime('2025-06-15T11:00:00.000-03:00')}]->(p);

MATCH (c:Cliente {email: 'augusto_carrara@hotmail.com'}), (p:Produto {sku: '00002'})
CREATE (c)-[:COMPROU {pedido_id: 2, data:
datetime('2025-07-04T18:00:00.000-03:00')}]->(p);

-- Interações de Gerson Flamengo
MATCH (c:Cliente {email: 'gerson_flamengo@yahoo.com'}), (p:Produto {sku: '00001'})
CREATE (c)-[:VISUALIZOU {data: datetime('2025-07-03T14:00:00.000-03:00')}]->(p);
MATCH (c:Cliente {email: 'gerson_flamengo@yahoo.com'}), (p:Produto {sku: '00003'})
CREATE (c)-[:AVALIOU {nota: 5, comentario: 'Excelente qualidade de som!', data:
datetime('2025-07-02T09:00:00.000-03:00')}]->(p);
```

3º Consultas:

```
-- Verificar os Nós existentes:
MATCH (n) RETURN n LIMIT 25;

-- Verificar os Nós e Relacionamentos:
MATCH p=()-[]->() RETURN p LIMIT 25;

-- Verifica os Clientes que compraram o "Notebook Dell'' também compraram quais outros
```

produtos?

```
MATCH (p1:Produto {sku: '00001'})<-[:COMPROU]-(cliente)-[:COMPROU]->(p2:Produto)
WHERE p1 <> p2
RETURN
    p2.nome AS ProdutoRecomendado,
    count(p2) AS FrequenciaDeCompraConjunta
ORDER BY
    FrequenciaDeCompraConjunta DESC;
```