

GRUPO 1 - KeroKomer

Diego Moura
Gisele Siqueira
Marcelo Barbugli
Matheus Higa
Ricardo Geroto
Roberto Eyama

Etapa 0: propor uma aplicação fictícia

O objetivo deste trabalho é aplicar os conceitos de Repositórios de dados e NoSQL, desenvolvendo um modelo de dados para uma aplicação fictícia de delivery de comida, o KeroKomer. Inspirado no iFood, o aplicativo reúne diversos restaurantes de diferentes tipos de cozinha, no qual os usuários podem escolher os produtos através dos cardápios dos restaurantes, fazer o pedido pelo aplicativo e receber o pedido em casa, entregue por um entregador.



Aplicativo KeroKomer delivery de comida

O grupo optou por usar o Apache Cassandra para desenvolver o aplicativo de delivery devido às suas características, pois é uma base de dados distribuída que oferece alta disponibilidade e tolerância a falhas, escalabilidade e desempenho consistente.

Principais características do o Apache Cassandra:

- alta disponibilidade e tolerância a falhas: replica dados em múltiplos datacenters.
- escalabilidade horizontal: adição fácil de nós ao cluster para lidar com o aumento da carga.
- desempenho de escrita: otimizado para operações de escrita rápidas, crucial para registros em tempo real.
- leituras rápidas: oferece leituras eficientes quando os dados são modelados corretamente.
- modelo de dados flexível: usa um modelo baseado em colunas, adaptável a diferentes necessidades de consulta.
- capacidade de big data: gerencia grandes volumes de dados, como logs de interação e dados de gps.
- configuração de consistência: permite ajustar o nível de consistência das operações de leitura e escrita.

Vantagens das Clustering Keys:

- ordenação de dados: organiza dados dentro de uma partição, útil para consultas ordenadas.
- consultas eficientes: otimiza consultas com filtragem e ordenação.
- agrupamento de dados: agrupa dados relacionados, facilitando a recuperação eficiente.
- flexibilidade na modelagem de dados: permite definir múltiplas clustering keys para suportar diferentes tipos de consultas.
- leituras otimizadas: reduz i/o e melhora a performance ao buscar apenas os dados necessários.
- implementação de paginação: facilita a paginação de resultados, mantendo a ordem dos dados.

Etapas 1: Criar o fluxo de trabalho do aplicativo

Levantamento de Requisitos do Domínio de Negócio

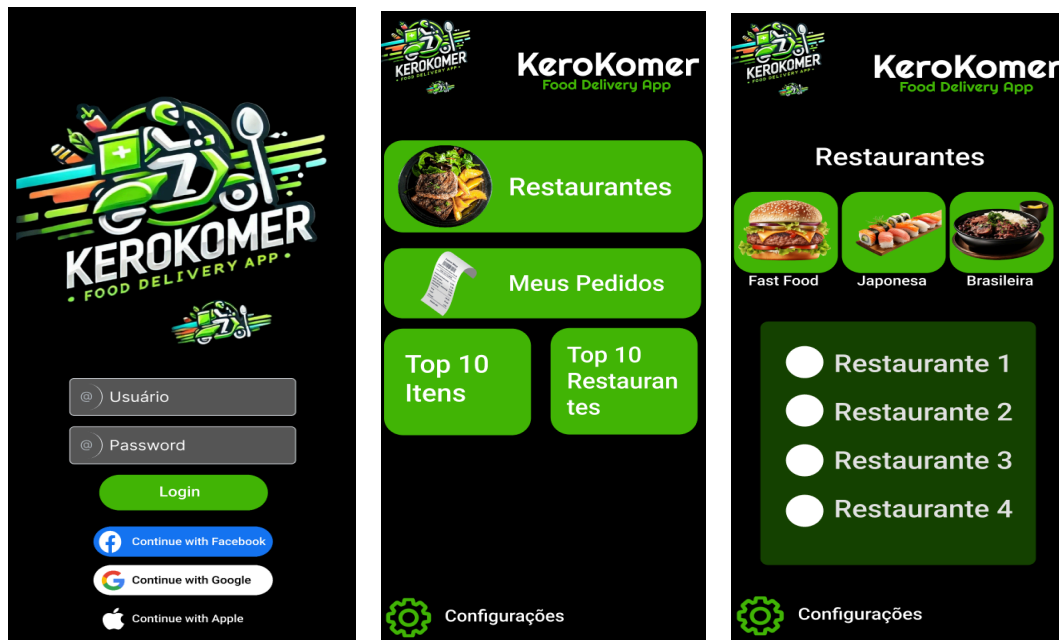
- Usuários: Podem se cadastrar, fazer login, procurar restaurantes, fazer pedidos, acompanhar o status dos pedidos, reclamação, avaliação de restaurante/entregador, acessar perfil dos restaurantes.
- Restaurantes: Podem se cadastrar, listar seus menus e gerar promoções, atualizar disponibilidade e gerenciar (aceitar / recusar / cancelar) pedidos.
- Entregadores: Podem se cadastrar, aceitar/negar pedidos para entrega e atualizar o status da entrega.

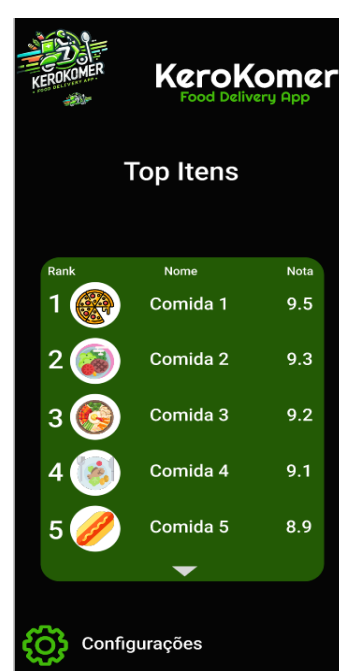
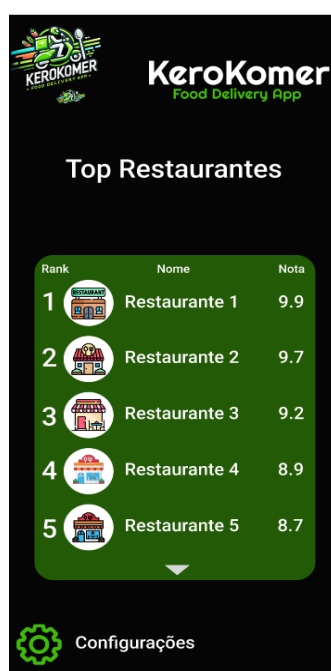
- Administradores: Podem gerenciar usuários, restaurantes e entregadores, gestão do processo.

Wireframe/Proof-of-Concept

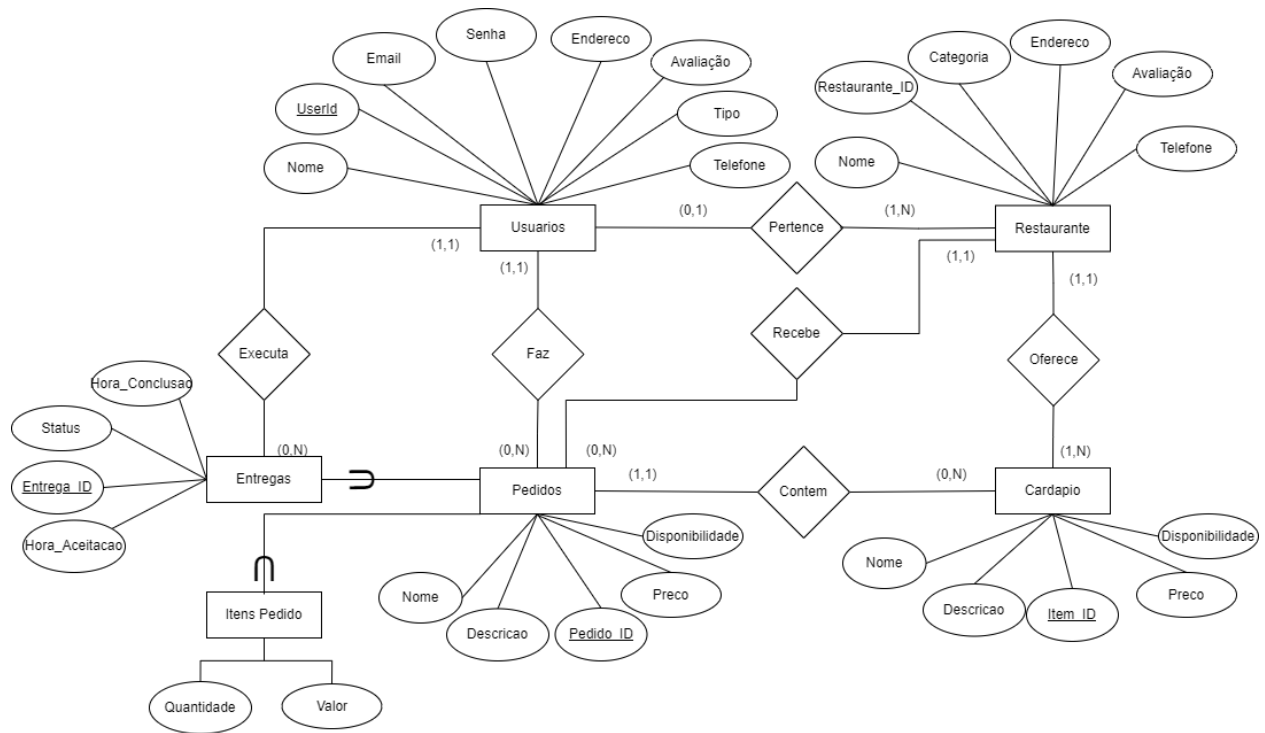
As principais telas são:

- Tela de Login/Cadastro: Para usuários, restaurantes e entregadores.
- Tela de Pesquisa de Restaurantes: Lista de restaurantes com filtros.
- Tela de Menu do Restaurante: Lista de itens disponíveis para pedido.
- Tela de Carrinho de Compras: Revisão e confirmação do pedido.
- Tela de Acompanhamento de Pedido: Status em tempo real do pedido.
- Tela de Recebimento de Pedido: Confirmação do pedido, avaliação do pedido/entrega, observação.
- Painel do Restaurante: Gerenciamento de menus e pedidos.
- Painel do Entregador: Aceitação e atualização do status de entrega.
- Painel do Administrador: Gerenciamento geral do sistema.

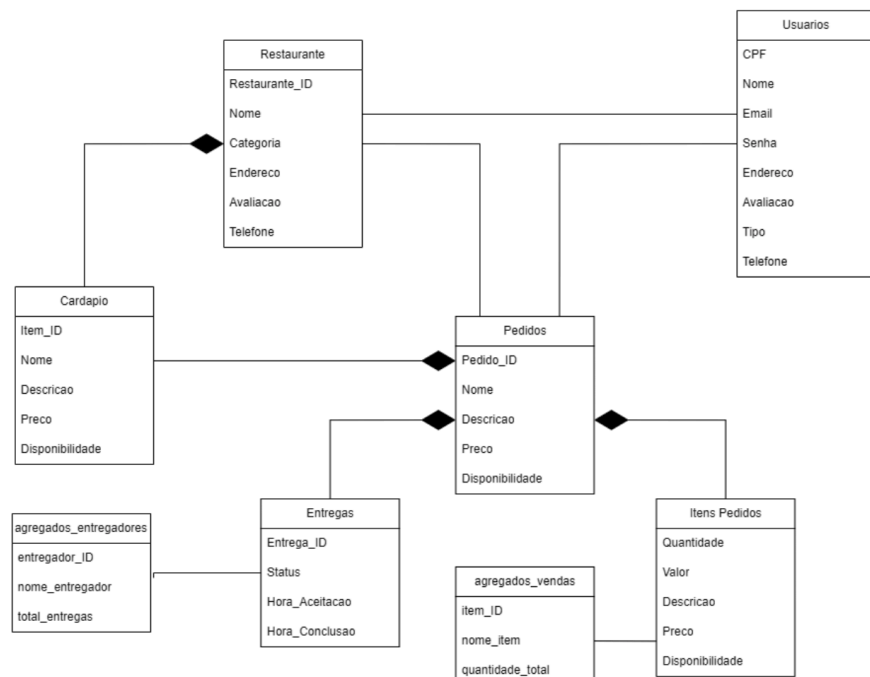




Modelo de dados Entidade Relacionamento

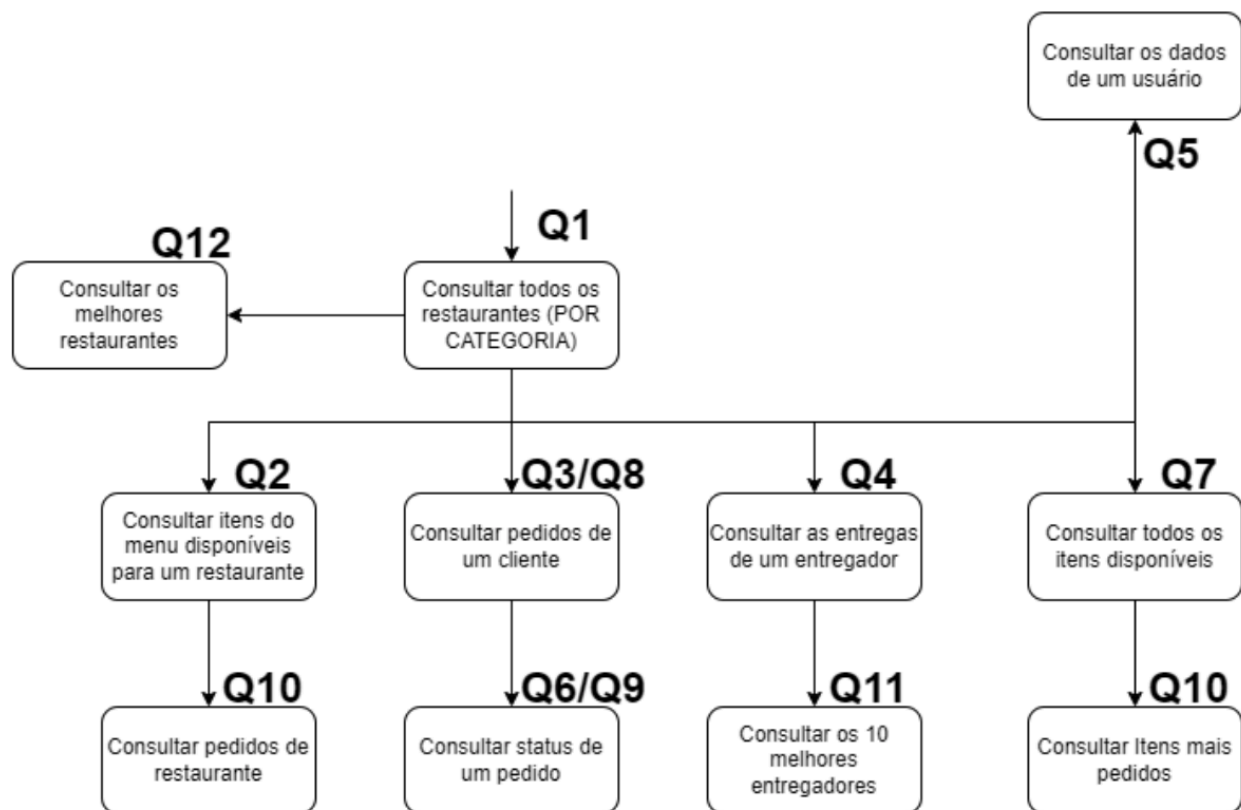


Modelagem de Agregados



Criação do modelo de Dados Lógicos

- Q1** - Consultar Restaurantes por `Categoria`
- Q2** - Consultar Itens de Menu de um Restaurante
- Q3** - Consultar `Pedidos` de um `Cliente`
- Q4** - Consultar Entregas finalizadas de um Entregador:
- Q5** - Consultar Informações de um `Usuário`
- Q6** - Consultar `Pedidos` por `Status`
- Q7** - Consultar Todos os `Itens` de Menu Disponíveis
- Q8** - Consultar Detalhes de um `Pedido` Específico
- Q9** - Consultar Pedidos de um Restaurante
- Q10** - Consultar Itens Mais Vendidos
- Q11** - Consulta para Top 10 Entregadores
- Q12** - Consulta para Top 5 Restaurantes



Modelo de Dados para Cassandra

Usuários (Tabela: usuarios)

Partition Key: CPF

Colunas: nome, email, senha, endereco, telefone, tipo, avaliacao

Restaurantes (Tabela: restaurantes)

Partition Key: restaurante_id

Colunas: nome, endereco, telefone, categoria, proprietario_id, avaliacao

Itens de Menu (Tabela: cardapio)

Partition Key: restaurante_id

Clustering Key: item_id

Colunas: nome, descricao, preco, disponibilidade

Pedidos (Tabela: pedidos)

Partition Key: pedido_id

Colunas: cliente_id, restaurante_id, data_hora_pedido, status, total

Coleção (Map): itens (map<text, int>) - item_id como chave e quantidade como valor

Entregas (Tabela: entregas)

Partition Key: entrega_id

Colunas: pedido_id, entregador_id, status, hora_aceitacao, hora_conclusao

Etapas 3: Criação das tabelas, população de dados e realização de consultas planejadas

Tabela usuarios

cql

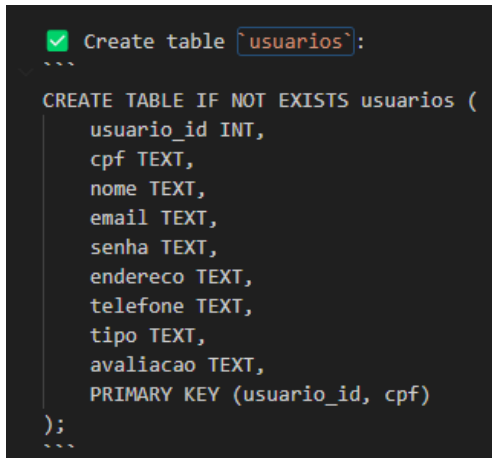
```
CREATE TABLE usuarios (  
    usuario_id UUID PRIMARY KEY,  
    CPF TEXT,  
    nome TEXT,  
    email TEXT,  
    senha TEXT,  
    endereco TEXT,  
    telefone TEXT,  
    tipo TEXT,  
    avaliacao TEXT  
);
```

Dados Fictícios:

cql

```
INSERT INTO usuarios (usuario_id, nome, email, senha, endereco,
telefone, tipo, cpf)
VALUES (uuid(), 'João Silva', 'joao@example.com',
'hashed_password', 'Rua A, 123', '123456789', 'Cliente',
'111.222.333-44');
```

Print da criação da tabela no Cassandra:

A screenshot of a terminal window with a dark background. At the top, a green checkmark icon is followed by the text 'Create table `usuarios`:' in a light blue font. Below this, the CQL command to create the 'usuarios' table is displayed in a light blue monospace font. The command includes fields for usuario_id (INT), cpf (TEXT), nome (TEXT), email (TEXT), senha (TEXT), endereco (TEXT), telefone (TEXT), tipo (TEXT), avaliacao (TEXT), and a primary key on (usuario_id, cpf). The command ends with a semicolon and is followed by three tilde characters (~ ~ ~).

```
✓ Create table `usuarios`:
~ ~ ~
CREATE TABLE IF NOT EXISTS usuarios (
  usuario_id INT,
  cpf TEXT,
  nome TEXT,
  email TEXT,
  senha TEXT,
  endereco TEXT,
  telefone TEXT,
  tipo TEXT,
  avaliacao TEXT,
  PRIMARY KEY (usuario_id, cpf)
);
~ ~ ~
```

Tabela restaurantes

cql

```
CREATE TABLE restaurantes (
  restaurante_id UUID PRIMARY KEY,
  nome TEXT,
  endereco TEXT,
  telefone TEXT,
  categoria TEXT,
  proprietario_id UUID
);
```

Dados Fictícios:

cql

```
INSERT INTO restaurantes (restaurante_id, nome, endereco,
telefone, categoria, proprietario_id)
VALUES (uuid(), 'Restaurante X', 'Avenida B, 456', '987654321',
'Italiano', uuid());
```


Print da criação da tabela no Cassandra:

```
✓ Create table `restaurantes`:  
...  
CREATE TABLE IF NOT EXISTS restaurantes (  
    restaurante_id INT,  
    nome TEXT,  
    endereco TEXT,  
    telefone TEXT,  
    categoria TEXT,  
    proprietario_id INT,  
    PRIMARY KEY (restaurante_id, proprietario_id)  
);  
...
```

Tabela cardapio

cql

```
CREATE TABLE cardapio(  
    restaurante_id UUID,  
    item_id UUID,  
    nome TEXT,  
    descricao TEXT,  
    preco DECIMAL,  
    disponibilidade BOOLEAN,  
    PRIMARY KEY (restaurante_id, item_id)  
);
```

Dados Fictícios:

cql

```
INSERT INTO cardapio (restaurante_id, item_id, nome, descricao,  
preco, disponibilidade)  
VALUES (uuid(), uuid(), 'Pizza Margherita', 'Pizza com tomate,  
mozzarella e manjeriç o', 25.99, true);
```

Print da criação da tabela no Cassandra:

```
✓ Create table `cardapio`:  
...  
CREATE TABLE IF NOT EXISTS cardapio(  
    item_id INT,  
    restaurante_id INT,  
    nome TEXT,  
    descricao TEXT,  
    preco DECIMAL,  
    disponibilidade BOOLEAN,  
    PRIMARY KEY (item_id, restaurante_id)  
);  
...
```

Tabela pedidos

cql

```
CREATE TABLE pedidos (  
    pedido_id UUID PRIMARY KEY,  
    cliente_id UUID,  
    restaurante_id UUID,  
    data_hora_pedido TIMESTAMP,  
    status TEXT,  
    total DECIMAL,  
    itens MAP<TEXT, INT>  
);
```

Dados Fictícios:

cql

```
INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id,  
data_hora_pedido, status, total, itens)  
VALUES (uuid(), uuid(), uuid(), '2024-06-19T12:00:00Z',  
'Pendente', 55.98, {'item1': 1, 'item2': 1});
```

Print da criação da tabela no Cassandra:

```
✓ Create table `pedidos`:  
CREATE TABLE pedidos (  
    pedido_id INT,  
    cliente_id INT,  
    restaurante_id INT,  
    data_hora_pedido TIMESTAMP,  
    status TEXT,  
    total DECIMAL,  
    itens MAP<TEXT, INT>,  
    PRIMARY KEY (pedido_id, cliente_id, restaurante_id)  
);
```

Tabela entregas

cql

```
CREATE TABLE entregas (  
    entrega_id UUID PRIMARY KEY,  
    pedido_id UUID,  
    entregador_id UUID,  
    status TEXT,  
    hora_aceitacao TIMESTAMP,  
    hora_conclusao TIMESTAMP  
);
```

Dados Fictícios:

cql

```
INSERT INTO entregas (entrega_id, pedido_id, entregador_id,  
status, hora_aceitacao, hora_conclusao)  
VALUES (uuid(), uuid(), uuid(), 'A Caminho',  
'2024-06-19T12:15:00Z', null);
```

Print da criação da tabela no Cassandra:

```
✓ Create table `entregas`:  
...  
CREATE TABLE IF NOT EXISTS entregas (  
    entrega_id INT,  
    pedido_id INT,  
    entregador_id INT,  
    status TEXT,  
    hora_aceitacao TIMESTAMP,  
    hora_conclusao TIMESTAMP,  
    PRIMARY KEY (entrega_id, pedido_id, entregador_id)  
);  
...
```

Tabelas de Agregados:

Adicionando a Tabela de Vendas de Itens

Vamos criar uma tabela vendas_itens que registra cada venda de item.

Tabela vendas_itens

cql

```
CREATE TABLE vendas_itens (  
    venda_id UUID PRIMARY KEY,  
    item_id UUID,  
    restaurante_id UUID,  
    quantidade INT,  
    data_venda TIMESTAMP  
);
```

Dados Fictícios:

cql

```
INSERT INTO vendas_itens (venda_id, item_id, restaurante_id,  
quantidade, data_venda)  
VALUES (uuid(), uuid(), uuid(), 3, '2024-06-19T12:00:00Z');  
INSERT INTO vendas_itens (venda_id, item_id, restaurante_id,  
quantidade, data_venda)  
VALUES (uuid(), uuid(), uuid(), 5, '2024-06-19T13:00:00Z');
```

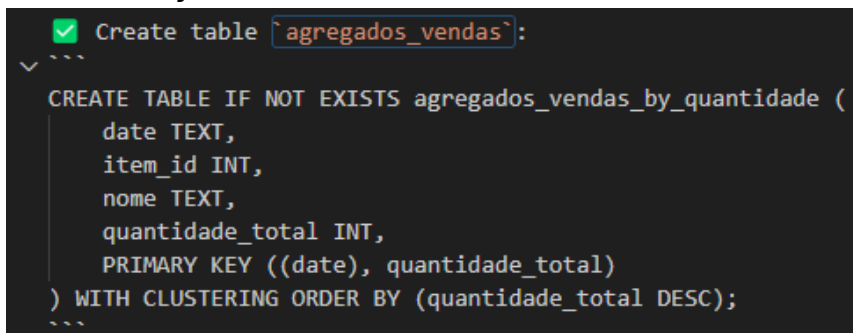
Tabela de Agregados de Vendas

```
cql
CREATE TABLE agregados_vendas (
    item_id UUID PRIMARY KEY,
    nome_item TEXT,
    quantidade_total INT
);
```

Dados Fictícios:

```
cql
INSERT INTO agregados_vendas (item_id, nome_item,
quantidade_total)
VALUES (uuid(), 'Pizza Margherita', 50);
INSERT INTO agregados_vendas (item_id, nome_item,
quantidade_total)
VALUES (uuid(), 'Pasta Carbonara', 30);
```

Print da criação da tabela no Cassandra:

A screenshot of a Cassandra query execution interface. At the top, a green checkmark icon is next to the text 'Create table `agregados_vendas`:'. Below this, a dropdown arrow points to a code block containing the following SQL statement:

```
CREATE TABLE IF NOT EXISTS agregados_vendas_by_quantidade (
    date TEXT,
    item_id INT,
    nome TEXT,
    quantidade_total INT,
    PRIMARY KEY ((date), quantidade_total)
) WITH CLUSTERING ORDER BY (quantidade_total DESC);
```

Procedimento de Atualização de Agregados

Para manter a tabela de agregados atualizada, você deve atualizar os agregados toda vez que um item é vendido.

```
cql
UPDATE agregados_vendas SET quantidade_total = quantidade_total
+ 5 WHERE item_id = uuid();
```

Tabela de Agregados de Entregas por Entregador

Para facilitar a consulta dos 10 melhores entregadores, podemos criar uma tabela de agregados que armazena o total de entregas feitas por cada entregador.

Tabela agregados_entregadores

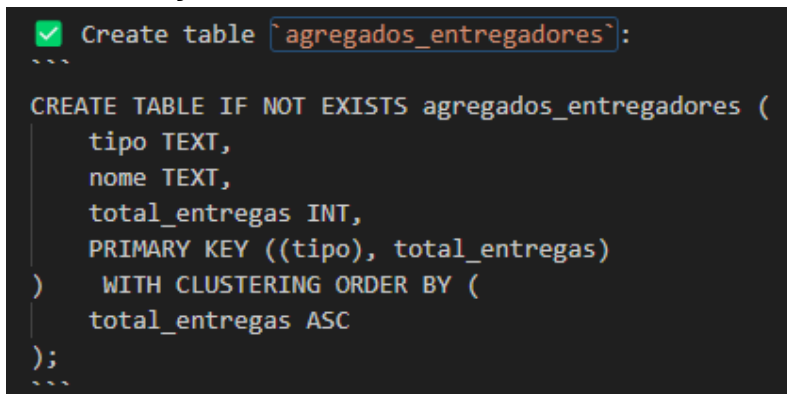
```
cql
CREATE TABLE agregados_entregadores (
    entregador_id UUID PRIMARY KEY,
    nome_entregador TEXT,
    total_entregas INT
);
```

Inserção de Dados Fictícios

Para alimentar a tabela com dados fictícios, podemos usar as seguintes inserções:

```
cql
INSERT INTO agregados_entregadores (entregador_id,
nome_entregador, total_entregas)
VALUES (uuid(), 'Carlos Mendes', 150);
INSERT INTO agregados_entregadores (entregador_id,
nome_entregador, total_entregas)
VALUES (uuid(), 'Ana Silva', 130);
```

Print da criação da tabela no Cassandra:



```
✓ Create table `agregados_entregadores`:
...
CREATE TABLE IF NOT EXISTS agregados_entregadores (
    tipo TEXT,
    nome TEXT,
    total_entregas INT,
    PRIMARY KEY ((tipo), total_entregas)
) WITH CLUSTERING ORDER BY (
    total_entregas ASC
);
...
```

Atualização da Tabela de Agregados

Para manter a tabela de agregados atualizada, sempre que uma entrega for concluída, atualizamos o contador:

```
cql
UPDATE agregados_entregadores
SET total_entregas = total_entregas + 1
WHERE entregador_id = uuid();
```

Tabela de Agregados de Avaliações dos Restaurantes

```
✓ Create table `agregados_avaliacoes_restaurantes`:  
...  
CREATE TABLE IF NOT EXISTS agregados_avaliacoes_restaurantes (  
    tipo TEXT,  
    restaurante_id INT,  
    nome TEXT,  
    avaliacao_media DECIMAL,  
    total_avaliacoes INT,  
    PRIMARY KEY ((tipo), avaliacao_media, total_avaliacoes),  
) WITH CLUSTERING ORDER BY (  
    avaliacao_media DESC,  
    total_avaliacoes DESC  
);  
...
```

População de Dados no Cassandra:

Programa para gerar massa de dados para teste das consultas.

Cria todos os 'INSERT' para que os dados sigam as estruturas definidas no modelo de dados e atendam aos requisitos da aplicação proposta.

```
eng-dados-pos > NoSQL > KeroKomer-final-project > app > insert_generation.py > generate_agregados_vendas_statement  
1  import os  
2  import random  
3  import string  
4  from datetime import datetime, timedelta  
5  from collections import Counter, defaultdict  
6  
7  # Funções gerais  
8  def generate_random_id(length=6):  
9      lower_bound = 10**(length-1)  
10     upper_bound = (10**length) - 1  
11     return random.randint(lower_bound, upper_bound)  
12  
13  def generate_random_string(length=8):  
14     letters = string.ascii_letters  
15     return ''.join(random.choice(letters) for i in range(length))  
16
```

Exemplo da função que cria dados para a tabela 'usuario':

```
# Gerar usuários
def generate_usuario_statements(num_records):
    statements = []
    user_ids = []
    entregador_ids = []

    tipo_quantidades = {
        'Restaurante': 10,
        'Entregador': 45,
        'Administrador': 10,
        'Usuario': num_records - 65
    }

    tipos = []
    for tipo, quantidade in tipo_quantidades.items():
        tipos.extend([tipo] * quantidade)

    random.shuffle(tipos)

    for i in range(1, num_records + 1):
        user_id = generate_random_id()
        user_ids.append(user_id)
        if tipos[i-1] == 'Entregador':
            entregador_ids.append(user_id)
            cpf = f'{10000000000 + i:011d}'
            nome = f'Nome{i} Sobrenome{i}'
            email = f'nome{i}.sobrenome{i}@example.com'
            senha = f'senha{i:03d}'
            endereco = f'Endereco {i}'
            telefone = f'{55250000000 + i:011d}'
            tipo = tipos[i-1]
            avaliacao = round(random.uniform(3, 5), 1) if tipo in ['Restaurante', 'Entregador'] else 'NULL'

            statement = f"INSERT INTO usuarios (user_ID, CPF, nome, email, senha, endereco, telefone, tipo, avaliacao) VALUES ('{user_id}', '{cpf}', '{nome}', '{email}', '{senha}', '{endereco}', '{telefone}', '{tipo}', '{avaliacao}');"
            statements.append(statement)

    return statements, user_ids, entregador_ids
```

O script a seguir unifica todos os dados que foram gerados em TXT e transforma em CQL

```
eng-dados-pos > NoSQL > KeroKomer-final-project > app > txt_to_cql.py > merge_txt_files_to_cql

1  import os
2
3  def merge_txt_files_to_cql(directory):
4      # Define o caminho do arquivo de saída
5      output_file_path = os.path.join(directory, "insert_statement.cql")
6
7      # Ordem específica dos arquivos
8      file_order = [
9          "usuario_insert_statements.txt",
10         "restaurante_insert_statements.txt",
11         "cardapio_insert_statements.txt",
12         "pedido_insert_statements.txt",
13         "entrega_insert_statements.txt",
14         "agregVendas_insert_statements.txt",
15         "agregRestaurante_insert_statements.txt",
16         "agregEntregadores_insert_statements.txt"
17     ]
18
19     # Abre o arquivo de saída no modo de escrita
20     with open(output_file_path, 'w', encoding='utf-8', errors='ignore') as output_file:
21         # Percorre os arquivos na ordem especificada
22         for filename in file_order:
23             file_path = os.path.join(directory, filename)
24             # Verifica se o arquivo existe no diretório
25             if os.path.isfile(file_path):
26                 # Abre o arquivo de entrada no modo de leitura
27                 try:
28                     with open(file_path, 'r', encoding='utf-8', errors='ignore') as input_file:
29                         # Lê todas as linhas do arquivo de entrada e escreve no arquivo de saída
30                         for line in input_file:
31                             output_file.write(line)
32                 except Exception as e:
33                     print(f"Erro ao ler o arquivo {file_path}: {e}")
34             else:
35                 print(f"Arquivo {file_path} não encontrado.")
36
37     print(f"Todos os arquivos .txt foram mesclados em {output_file_path}")
38
```


Exemplo do resultado

```
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 745, 'Item 745', 9);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 473, 'Item 473', 18);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 511, 'Item 511', 5);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 894, 'Item 894', 21);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 971, 'Item 971', 9);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 840, 'Item 840', 29);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 301, 'Item 301', 15);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 101, 'Item 101', 15);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 733, 'Item 733', 2);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 260, 'Item 260', 1);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 285, 'Item 285', 5);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 737, 'Item 737', 8);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 355, 'Item 355', 9);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 887, 'Item 887', 6);
INSERT INTO agregados_vendas_by_quantidade (date, item_id, nome, quantidade_total) VALUES ('2024-01-30', 162, 'Item 162', 1);
```

Tabela preenchida com os dados de usuários

```
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (67063503, 116409, 365697,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (39737219, 495802, 465096,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (87715119, 839076, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (73230038, 908532, 281595,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (84835251, 267480, 916251,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (65057972, 119178, 365697,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (93604879, 288769, 543884,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (94919140, 162986, 772186,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (22128529, 453010, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (29910148, 685503, 465096,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (41490675, 606778, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (96438404, 125611, 625762,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (38710062, 867704, 625762,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (73403028, 662286, 465096,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (16928463, 923706, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (86576357, 699062, 916251,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (11740090, 758958, 450666,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (70492669, 921086, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (27523458, 671132, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (31987061, 420911, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (21946881, 274065, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (59797577, 871046, 281595,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (48380602, 109315, 610876,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (37614017, 751800, 450666,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (53561891, 117521, 772186,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (50331196, 605688, 281595,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (34345053, 247912, 281595,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (33653847, 583110, 610876,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (42944419, 528431, 543884,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (21764812, 293555, 450666,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (51209493, 830561, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (89437912, 993097, 450666,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (11543986, 108867, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (64792164, 445758, 872427,
qqlsh:kerokomer_key> INSERT INTO pedidos (pedido_id, cliente_id, restaurante_id, data_hora_pedido, status, total, itens) VALUES (66065966, 236413, 772186,
```

Consultas Correspondentes

1. Consultar Restaurantes por Categoria

cql

```
SELECT * FROM restaurantes WHERE categoria = 'Brasileira'  
ALLOW FILTERING;
```

```
☑ Consultar Restaurantes por `Categoria`:  
~~~  
SELECT * FROM restaurantes WHERE categoria = 'Brasileira' ALLOW FILTERING;  
~~~
```

```
cqlsh:kerokomer_key> SELECT * FROM restaurantes WHERE categoria = 'Brasileira' ALLOW FILTERING;
```

restaurante_id	categoria	endereço	nome	proprietario_id	telefone
610876	Brasileira	Endereo 8	Restaurante 8	569029	1100000000008
872427	Brasileira	Endereo 5	Restaurante 5	550993	1100000000005
543884	Brasileira	Endereo 4	Restaurante 4	108867	1100000000004
281595	Brasileira	Endereo 3	Restaurante 3	117838	1100000000003
916251	Brasileira	Endereo 6	Restaurante 6	347027	1100000000006

(5 rows)

2. Consultar Itens de Menu de um Restaurante

cql

```
SELECT * FROM cardapio WHERE restaurante_id = uuid();
```

```
☑ Consultar Itens de Menu de um Restaurante:  
~~~  
SELECT * FROM cardapio WHERE restaurante_id = 543884;  
~~~
```

```
cqlsh:kerokomer_key> SELECT * FROM cardapio WHERE restaurante_id = (543884);
```

restaurante_id	item_id	descricao	disponibilidade	nome	preco
543884	977	Descricao do item 163	True	Item 163	47.14
543884	948	Descricao do item 131	False	Item 131	18.5
543884	890	Descricao do item 60	True	Item 60	99.49
543884	880	Descricao do item 116	False	Item 116	44.61
543884	871	Descricao do item 100	True	Item 100	26.53
543884	851	Descricao do item 103	True	Item 103	60.08
543884	833	Descricao do item 179	True	Item 179	69.48
543884	797	Descricao do item 177	True	Item 177	68.56
543884	758	Descricao do item 30	True	Item 30	63.86
543884	711	Descricao do item 19	False	Item 19	62.28
543884	541	Descricao do item 180	True	Item 180	75.43
543884	526	Descricao do item 4	True	Item 4	98.07
543884	493	Descricao do item 149	False	Item 149	48.37
543884	411	Descricao do item 77	True	Item 77	45.29
543884	404	Descricao do item 135	True	Item 135	19.1
543884	390	Descricao do item 191	True	Item 191	18.53
543884	388	Descricao do item 90	True	Item 90	31.02
543884	370	Descricao do item 170	True	Item 170	69.33
543884	364	Descricao do item 79	True	Item 79	68.48
543884	336	Descricao do item 141	True	Item 141	24.74
543884	247	Descricao do item 8	True	Item 8	31.42
543884	178	Descricao do item 14	True	Item 14	93.77
543884	137	Descricao do item 185	True	Item 185	43.21
543884	131	Descricao do item 162	True	Item 162	62.29
543884	120	Descricao do item 110	True	Item 110	75.82

(25 rows)

3. Consultar Pedidos de um Cliente

cql

```
SELECT * FROM pedidos WHERE cliente_id = uuid() ALLOW FILTERING;
```

✓ Consultar `Pedidos` de um `Cliente`:

~~~

```
SELECT * FROM pedidos WHERE cliente_id = (922304) ALLOW FILTERING;
```

~~~

```
cqlsh:kerokomer_key> SELECT * FROM pedidos WHERE cliente_id = (922304) ALLOW FILTERING;
```

pedido_id	cliente_id	data_hora_pedido	itens	restaurante_id	status	total
25639771	922304	2024-04-12 12:40:02.579000+0000	{'137': 1}	543884	Cancelled	43.21

(1 rows)

4. Consultar Entregas em Finalizadas de um Entregador

cql

```
SELECT * FROM entregas WHERE entregador_id = uuid() AND status = 'Completed' ALLOW FILTERING;
```

✓ Consultar Entregas finalizadas de um Entregador:

~~~

```
SELECT entrega_id, pedido_id, entregador_id, status FROM entregas WHERE entregador_id = (384201) AND status = 'Completed' ALLOW FILTERING;
```

~~~

```
cqlsh:kerokomer_key> SELECT entrega_id, pedido_id, entregador_id, status FROM entregas WHERE entregador_id = (384201) AND status = 'Completed' ALLOW FILTERING;
```

entrega_id	pedido_id	entregador_id	status
87252148	24788905	384201	Completed
31772480	42587855	384201	Completed

(2 rows)

5. Consultar Informações de um Usuário

cql

```
SELECT * FROM usuarios WHERE usuario_id = uuid();
```

✓ Consultar Informações de um `Usuário`:

~~~

```
SELECT * FROM usuarios WHERE usuario_id = (125611);
```

~~~

```
cqlsh:kerokomer_key> SELECT * FROM usuarios WHERE usuario_id = (125611);
```

usuario_id	avaliacao	cpf	email	endereço	nome	senha	telefone	tipo
125611	null	10000000042	nome42.sobrenome42@example.com	Endereço 42	Nome42 Sobrenome42	senha042	55250000042	Usuario

(1 rows)

6. Consultar Pedidos por Status

cql

```
SELECT * FROM pedidos WHERE status = 'Pendente' ALLOW  
FILTERING;
```

```
☑ Consultar `Pedidos` por `Status`:  
~~~  
SELECT * FROM pedidos WHERE status = 'Pending' ALLOW FILTERING;  
~~~
```

```
cqlsh:kerokomer_key> SELECT * FROM pedidos WHERE status = 'Pending' ALLOW FILTERING;
```

pedido_id	cliente_id	data_hora_pedido	itens	restaurante_id	status	total
53425731	871731	2023-10-02 22:48:02.579000+0000	{'239': 7}	610876	Pending	393.68
94528454	652519	2023-07-14 05:06:02.579000+0000	{'617': 10}	872427	Pending	852.6999999999999
38710062	867704	2024-01-01 12:27:02.579000+0000	{'526': 1}	625762	Pending	42.41
76289072	649835	2024-06-02 15:49:02.579000+0000	{'541': 9}	543884	Pending	678.8700000000001
28181579	749309	2024-02-04 04:42:02.579000+0000	{'753': 10}	916251	Pending	941.5999999999999
68752893	560658	2024-02-20 05:00:02.579000+0000	{'450': 9}	872427	Pending	315.45
52734689	943171	2023-09-06 17:36:02.579000+0000	{'477': 2}	450666	Pending	26.86
57497373	171300	2023-09-15 03:42:02.579000+0000	{'493': 8}	543884	Pending	386.96
23517847	913705	2024-01-17 08:54:02.579000+0000	{'695': 10}	872427	Pending	996.9
43920143	850794	2024-05-07 23:42:02.579000+0000	{'765': 6}	450666	Pending	140.76
85719124	882934	2023-11-09 08:02:02.579000+0000	{'169': 2}	610876	Pending	21.92
27523458	671132	2023-07-31 16:30:02.579000+0000	{'134': 3}	872427	Pending	200.91
31706288	171080	2024-04-19 21:58:02.579000+0000	{'180': 9}	281595	Pending	491.13
87715119	839076	2023-08-09 00:42:02.579000+0000	{'134': 10}	872427	Pending	669.7
61038713	946087	2024-06-05 04:41:02.579000+0000	{'671': 9}	772186	Pending	706.9499999999999
99690704	265332	2023-07-01 05:20:02.579000+0000	{'430': 7}	772186	Pending	451.01000000000005
54531418	821001	2023-08-04 02:00:02.579000+0000	{'494': 8}	772186	Pending	751.6
34817536	770724	2023-12-08 04:56:02.579000+0000	{'530': 7}	625762	Pending	203.91
45640371	991555	2023-12-13 00:57:02.579000+0000	{'419': 1}	610876	Pending	75.42
75926512	326975	2024-05-12 02:17:02.579000+0000	{'424': 2}	625762	Pending	135.54
92456361	107174	2023-12-01 00:55:02.579000+0000	{'421': 7}	365697	Pending	595.2800000000001
56971724	457199	2023-08-21 02:45:02.579000+0000	{'178': 4}	543884	Pending	375.08
48080015	347027	2023-07-01 00:53:02.579000+0000	{'608': 4}	465096	Pending	274.08
51209493	830561	2023-11-18 15:17:02.579000+0000	{'164': 1}	872427	Pending	28.27
89437912	993097	2023-08-25 13:23:02.579000+0000	{'582': 10}	450666	Pending	758.7
90203960	476479	2024-06-15 11:25:02.579000+0000	{'120': 2}	543884	Pending	151.64
66988171	401930	2024-05-16 22:38:02.579000+0000	{'628': 3}	365697	Pending	176.34
38849679	882859	2024-05-06 16:38:02.579000+0000	{'130': 3}	365697	Pending	170.7
31050876	805180	2023-07-03 01:35:02.579000+0000	{'950': 3}	772186	Pending	247.20000000000002
91269765	513167	2024-01-11 10:22:02.579000+0000	{'871': 7}	543884	Pending	185.71
57070563	331112	2024-05-28 08:32:02.579000+0000	{'744': 9}	916251	Pending	330.93
91290695	211146	2023-11-30 06:03:02.579000+0000	{'634': 9}	450666	Pending	197.28000000000003

7. Consultar Todos os Itens de Menu Disponíveis

cql

```
SELECT * FROM cardapio WHERE disponibilidade = true ALLOW  
FILTERING;
```

☒ Consultar Todos os `Itens` de Menu Disponíveis:

```
SELECT * FROM cardapio WHERE disponibilidade = true ALLOW FILTERING;
```

```
cqlsh:kerokomer_key> SELECT * FROM cardapio WHERE disponibilidade = true ALLOW FILTERING
```

restaurante_id	item_id	descricao	disponibilidade	nome	preco
772186	962	Descricao do item 117	True	Item 117	25.68
772186	950	Descricao do item 26	True	Item 26	82.4
772186	729	Descricao do item 91	True	Item 91	41.99
772186	721	Descricao do item 115	True	Item 115	19.91
772186	688	Descricao do item 190	True	Item 190	69.07
772186	671	Descricao do item 119	True	Item 119	78.55
772186	651	Descricao do item 176	True	Item 176	92.59
772186	494	Descricao do item 22	True	Item 22	93.95
772186	454	Descricao do item 105	True	Item 105	32.41
772186	371	Descricao do item 49	True	Item 49	73.85
772186	346	Descricao do item 142	True	Item 142	32.4
365697	893	Descricao do item 172	True	Item 172	26.33
365697	887	Descricao do item 168	True	Item 168	20.44
365697	801	Descricao do item 124	True	Item 124	50.7
365697	781	Descricao do item 24	True	Item 24	58.98
365697	491	Descricao do item 112	True	Item 112	57.04
365697	421	Descricao do item 32	True	Item 32	85.04
365697	400	Descricao do item 12	True	Item 12	41.93
365697	346	Descricao do item 40	True	Item 40	19.31
365697	316	Descricao do item 101	True	Item 101	57.31
365697	297	Descricao do item 198	True	Item 198	84.83
365697	231	Descricao do item 189	True	Item 189	63.25
365697	214	Descricao do item 104	True	Item 104	65.91
365697	130	Descricao do item 59	True	Item 59	56.9
465096	906	Descricao do item 166	True	Item 166	62.33
465096	811	Descricao do item 182	True	Item 182	57.47
465096	676	Descricao do item 3	True	Item 3	20.07
465096	544	Descricao do item 36	True	Item 36	82.95
465096	351	Descricao do item 156	True	Item 156	36.42
465096	268	Descricao do item 92	True	Item 92	27.92
610876	934	Descricao do item 158	True	Item 158	83.26
610876	867	Descricao do item 75	True	Item 75	76.33

8. Consultar Detalhes de um Pedido Específico

cql

```
SELECT * FROM pedidos WHERE pedido_id = uuid();
```

☒ Consultar Detalhes de um `Pedido` Específico:

```
SELECT * FROM pedidos WHERE pedido_id = (78435300);
```

```
cqlsh:kerokomer_key> SELECT * FROM pedidos WHERE pedido_id = (78435300);
```

pedido_id	cliente_id	data_hora_pedido	itens	restaurante_id	status	total
78435300	578933	2024-01-16 23:02:02.579000+0000	{'477': 10}	450666	Cancelled	134.3

(1 rows)

9. Consultar Usuário por CPF

cql

```
SELECT * FROM usuarios WHERE cpf = '111.222.333-44' ALLOW  
FILTERING;
```

☒ Consultar Usuário por `CPF`:

```
SELECT * FROM usuarios WHERE cpf = '1000000025' ALLOW FILTERING;
```

```
cqlsh:kerokomer_key> SELECT * FROM usuarios WHERE cpf = '1000000025' ALLOW FILTERING;
```

usuario_id	avaliacao	cpf	email	endereco	nome	senha	telefone	tipo
698676	null	1000000025	nome25.sobrenome25@example.com	Endereco 25	Nome25 Sobrenome25	senha025	55250000025	Usuario

(1 rows)

10. Consultar Pedidos de um Restaurante

cql

```
SELECT * FROM pedidos WHERE restaurante_id = uuid() ALLOW FILTERING;
```

☒ Consultar Pedidos de um Restaurante:

```
SELECT * FROM pedidos WHERE restaurante_id = (872427) ALLOW FILTERING;
```

```
cqlsh:kerokomer_key> SELECT * FROM pedidos WHERE restaurante_id = (872427) ALLOW FILTERING;
```

pedido_id	cliente_id	data_hora_pedido	itens	restaurante_id	status	total
94528454	652519	2023-07-14 05:06:02.579000+0000	{'617': 10}	872427	Pending	852.6999999999999
25364460	404521	2024-02-01 09:06:02.579000+0000	{'621': 10}	872427	In Progress	553.4000000000001
68752893	560658	2024-02-20 05:00:02.579000+0000	{'450': 9}	872427	Pending	315.45
23517847	913705	2024-01-17 08:54:02.579000+0000	{'695': 10}	872427	Pending	996.9
40087515	616227	2023-10-14 12:59:02.579000+0000	{'665': 1}	872427	Completed	47.3
27523458	671132	2023-07-31 16:30:02.579000+0000	{'134': 3}	872427	Pending	200.91
21946881	274065	2024-03-04 20:37:02.579000+0000	{'469': 4}	872427	In Progress	337.52
87715119	839076	2023-08-09 00:42:02.579000+0000	{'134': 10}	872427	Pending	669.7
59415366	521232	2024-02-15 03:55:02.579000+0000	{'469': 1}	872427	Completed	84.38
20319577	294006	2023-09-10 08:58:02.579000+0000	{'744': 1}	872427	In Progress	94.09
57122146	625058	2024-06-21 05:17:02.579000+0000	{'851': 7}	872427	In Progress	535.0799999999999
33934973	735513	2024-01-17 17:07:02.579000+0000	{'665': 10}	872427	In Progress	473.0
30006980	132425	2023-07-10 03:52:02.579000+0000	{'695': 7}	872427	In Progress	697.8299999999999
51209493	830561	2023-11-18 15:17:02.579000+0000	{'164': 1}	872427	Pending	28.27
21298251	935266	2024-03-02 18:03:02.579000+0000	{'387': 8}	872427	Cancelled	197.04
22128529	453010	2023-12-05 02:30:02.579000+0000	{'282': 5}	872427	In Progress	425.85
70702633	385644	2024-01-09 00:02:02.579000+0000	{'981': 8}	872427	Completed	407.36
31061576	425499	2024-05-22 14:49:02.579000+0000	{'716': 10}	872427	Completed	874.5999999999999
98564895	222147	2023-07-22 12:49:02.579000+0000	{'679': 10}	872427	Pending	612.9
80206261	706931	2023-06-26 17:01:02.579000+0000	{'981': 9}	872427	Cancelled	458.2800000000003
22187928	449610	2024-01-03 19:29:02.579000+0000	{'134': 9}	872427	Cancelled	602.73
25999261	278409	2024-02-01 15:36:02.579000+0000	{'220': 5}	872427	Pending	403.54999999999995
64547490	111845	2023-11-10 01:36:02.579000+0000	{'744': 2}	872427	In Progress	188.18
40670630	536279	2023-07-12 11:18:02.579000+0000	{'164': 3}	872427	Completed	84.81
55228617	220641	2023-09-14 08:47:02.579000+0000	{'147': 4}	872427	Pending	120.04
11543986	108867	2023-07-14 07:07:02.579000+0000	{'621': 1}	872427	Completed	55.34
16928463	923706	2024-04-27 11:26:02.579000+0000	{'645': 9}	872427	Completed	667.53
31987061	420911	2024-03-03 12:45:02.579000+0000	{'220': 3}	872427	In Progress	242.13
64792164	445758	2023-08-20 10:56:02.579000+0000	{'212': 4}	872427	Pending	63.04
33675291	117803	2024-04-07 01:17:02.579000+0000	{'591': 4}	872427	In Progress	212.6
70492669	921086	2023-11-29 07:37:02.579000+0000	{'447': 10}	872427	Pending	671.8000000000001
93574945	499152	2024-02-13 05:10:02.579000+0000	{'212': 1}	872427	Completed	15.76

Consultas nas tabelas de dados agregados:

11. Consultar Itens Mais Vendidos

Cql

Para realizar a consulta de itens mais vendidos, podemos utilizar a função de agregação **SUM** para somar a quantidade vendida de cada item e ordenar os resultados.

```
SELECT * FROM agregados_vendas ORDER BY
quantidade_total DESC;
```

☒ Consultar Itens Mais Vendidos:

```
SELECT * FROM agregados_vendas_by_quantidade WHERE date = '2024-01-30' limit 10;
```

```
cqlsh:kerokomer_key> SELECT * FROM agregados_vendas_by_quantidade WHERE date = '2024-01-30' limit 10;
```

date	quantidade_total	item_id	nome
2024-01-30	35	648	Item 648
2024-01-30	29	840	Item 840
2024-01-30	28	316	Item 316
2024-01-30	27	476	Item 476
2024-01-30	21	894	Item 894
2024-01-30	18	473	Item 473
2024-01-30	16	199	Item 199
2024-01-30	15	464	Item 464
2024-01-30	14	289	Item 289
2024-01-30	13	898	Item 898

12. Consulta para Top 10 Entregadores

cql

```
SELECT * FROM agregados_entregadores ORDER BY
total_entregas DESC LIMIT 10;
```

☒ Consulta para Top 10 Entregadores:

```
SELECT * FROM agregados_entregadores WHERE tipo = 'entregador' ORDER BY total_entregas DESC
ALLOW FILTERING;
```

```
cqlsh:kerokomer_key> SELECT * FROM agregados_entregadores WHERE tipo = 'entregador' ORDER BY total_entregas DESC ALLOW FILTERING;
```

tipo	total_entregas	nome
entregador	5	Nome211622 Sobrenome211622
entregador	4	Nome688316 Sobrenome688316
entregador	3	Nome786626 Sobrenome786626
entregador	2	Nome379052 Sobrenome379052
entregador	1	Nome572813 Sobrenome572813
entregador	0	Nome520365 Sobrenome520365

(6 rows)

13. Consulta para Top 10 Restaurantes

cql

```
SELECT * FROM agregados_avaliacoes_restaurantes ORDER BY  
avaliacao_media DESC LIMIT 10;
```

☒ Consulta para Top 10 Restaurantes:

```
SELECT * FROM agregados_avaliacoes_restaurantes WHERE tipo = 'restaurante' ORDER BY  
avaliacao_media, total_avaliacoes LIMIT 5 ALLOW FILTERING;
```

```
cqlsh:kerokomer_key> SELECT * FROM agregados_avaliacoes_restaurantes WHERE tipo = 'restaurante' LIMIT 5 ALLOW FILTERING;
```

tipo	avaliacao_media	total_avaliacoes	nome	restaurante_id
restaurante	4.9	139	Restaurante 10	311147
restaurante	4.8	164	Restaurante 3	228525
restaurante	4.8	29	Restaurante 7	498355
restaurante	4.7	292	Restaurante 2	495774
restaurante	4.5	432	Restaurante 4	552621

(5 rows)

```
cqlsh:kerokomer_key> █
```

Conclusão

O grupo desenvolveu um aplicativo fictício de entrega de comida, seguindo o modelo do iFood, aplicando os conhecimentos adquiridos ao decorrer da disciplina "Repositórios de Dados e NoSQL". Para isso, utilizamos o Apache Cassandra como banco de dados, por suas vantagens para sistemas distribuídos, como alta disponibilidade, escalabilidade horizontal, e desempenho otimizado para operações de escrita e leitura.

Através dele, conseguimos criar tabelas que suportam grandes volumes de dados e realizar consultas eficientes. A implementação das clustering keys foi fundamental para organizar, ordenar e agrupar dados, proporcionando consultas rápidas e eficientes, além de facilitar a paginação de resultados, como listas de pedidos.

Sendo assim, o uso do Apache Cassandra em nosso projeto demonstrou ser uma escolha robusta e eficaz, alinhada com os requisitos de desempenho, escalabilidade e disponibilidade de um aplicativo de entrega de comida.