UNIVERSIDADE TIRADENTES

PÓS-GRADUAÇÃO LATO SENSO

AVALIAÇÃO — BANCOS DE DADOS RELACIONAIS, NÃO RELACIONAIS E TÓPICOS AVANÇADOS - Estudos de Caso – Modelagem e Implementação

Prof. Msc. Thiago Lima

Dezembro de 2025

1. Instruções Gerais da Avaliação

Esta avaliação contém **4 estudos de caso**, cada um envolvendo a modelagem de dados em um banco de dados não relacional ou avançado.

O trabalho **pode ser feito individualmente ou em dupla**.

Cada estudo de caso vale **2,5 pontos**, totalizando **10 pontos**.

Prazo de entrega: **14/12/2025.**

Local da entrega: **Google Sala de Aula.**

1.1 Objetivos da Avaliação

- Avaliar a capacidade de modelar sistemas utilizando bancos não relacionais e avançados.
- Compreender diferenças entre modelos orientados a documentos, colunar, grafos e in-memory.
- Demonstrar domínio prático na construção de estruturas de dados eficientes.

1.2 Itens obrigatórios da entrega

A entrega final deverá conter:

1. **Todos os fontes produzidos**, incluindo:
    - Scripts de criação
    - Consultas
    - Arquivos JSON, CQL, Cypher, Redis etc.
    - Códigos auxiliares (se houver)
2. **Um vídeo explicativo** (entre 5 e 12 minutos) **para cada um dos 4 estudos de caso**, contendo:
    - Apresentação da modelagem
    - Demonstração da solução criada
    - Justificativas técnicas

1.3 Critérios de Avaliação

- Correção técnica
- Clareza e organização
- Justificativas conceituais
- Adequação da solução ao cenário
- Qualidade do vídeo e fontes enviados

2. Estudo de Caso 1 — Modelagem de Dados com MongoDB

🎯 1. Objetivos do Estudo de Caso

O sistema deve permitir:

1. Gerenciar **usuários (clientes e vendedores)**;
2. Cadastrar e consultar **produtos** com atributos variáveis;
3. Controlar **pedidos e histórico de compras**;
4. Registrar **avaliações de produtos**;
5. Garantir **escalabilidade, consistência e performance**.

🧠 2. Princípios de Modelagem no MongoDB

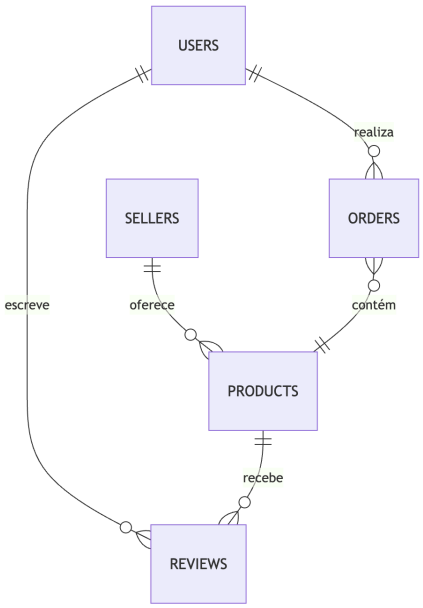
🔷 Documentos e Coleções

- Cada **documento** é uma unidade independente (JSON/BSON);
- As **coleções** agrupam documentos com estrutura semelhante;
- A modelagem é guiada pelos **padrões de acesso** (*query-driven design*).

🔷 Estratégias de Modelagem

- **Embedding (inclusão):** incorporar subdocumentos dentro de um documento pai.
  ➜ Ideal para dados acessados juntos com frequência.
- **Referencing (referência):** relacionar documentos via `_id`.
  ➜ Ideal para dados compartilhados entre várias entidades.

🗺️ 4. Diagrama Conceitual Simplificado



Tarefas (Responder)

1. Propor o **schema JSON** das coleções users, products e orders.
2. Justificar o uso de **embedding** e **referencing** no projeto.
3. Explicar como modelar produtos com atributos variáveis.
4. Criar um pequeno diagrama representando as coleções e relações.

3. Estudo de Caso 2 — Modelagem de Dados em Cassandra

Tema: Sistema de Gerenciamento de Pedidos para um Marketplace

---

🧭 Contexto do Problema

O marketplace precisa de um banco de dados capaz de:

- Escalar horizontalmente (milhares de usuários e produtos);
- Manter consultas rápidas sobre pedidos, produtos e histórico de clientes;
- Gerenciar dados altamente variáveis sem impactar a performance.

Cassandra é ideal para **cenários de leitura/escrita intensiva** com consistência eventual, sendo um banco **colunar distribuído**, que prioriza:

- Escrita rápida;
- Consultas eficientes para padrões predefinidos;
- Escalabilidade linear.

🎯 Objetivos da Modelagem

- Representar usuários, vendedores, produtos e pedidos;
- Otimizar consultas frequentes (busca por produtos, histórico de pedidos);
- Manter integridade eventual e histórico imutável;

- Permitir **particionamento (partition key)** eficiente para grandes volumes.

🧩 Tabelas Principais

Tabela	Objetivo	Partição	Clustering	Observações
`users`	Armazena clientes	`user_id`	-	Consulta rápida por cliente
`sellers`	Informações do vendedor	`seller_id`	-	Ranking e reputação
`products`	Catálogo de produtos	`category`	`product_id`	Consultas por categoria e ordenação
`orders_by_user`	Histórico de pedidos de um usuário	`user_id`	`order_date DESC`	Embedding dos itens do pedido
`reviews_by_product`	Avaliações de produto	`product_id`	`created_at DESC`	Histórico de reviews, ordenado por data

Em Cassandra, o design é **query-driven**, ou seja, modelamos as tabelas pensando **nas consultas que serão feitas**.

📦 Modelagem de Cada Tabela

1️⃣ Tabela `users`

Contém as informações cadastrais e preferências dos clientes.

- **Partition Key:** `user_id`
- **Exemplo de consulta:** histórico de pedidos por usuário

2️⃣ Tabela `sellers`

Armazena dados das lojas e seus indicadores de reputação.

- **Partition Key:** `seller_id`
- **Exemplo de consulta:** ranking de vendedores por nota

3️⃣ Tabela `products`

Contém os produtos disponíveis no marketplace.

- **Partition Key:** `category`
- **Clustering Key:** `product_id`
- **Objetivo:** consultas rápidas por categoria e ordenação de produtos

4️⃣ Tabela `orders_by_user`

Representa pedidos realizados pelos usuários.

- **Partition Key:** `user_id`
- **Clustering Key:** `order_date DESC`, `order_id`
- **Objetivo:** histórico de pedidos do usuário, ordenado pelo mais recente
- **Estratégia:** embedding dos itens do pedido para performance

5️⃣ Tabela `reviews_by_product`

Guarda avaliações dos usuários sobre produtos.

- **Partition Key:** `product_id`
- **Clustering Key:** `created_at DESC`, `user_id`
- **Objetivo:** histórico de avaliações, ordenado por data

⚙️ Consultas Comuns (Query-Driven)

- Buscar produtos de uma categoria:

```
SELECT * FROM products WHERE category='Eletrônicos' LIMIT 20;
```

Tarefas (Responder)

1. Explicar por que Cassandra exige **query-driven design**.
2. Desenhar a tabela orders_by_user completa.
3. Listar três consultas comuns e justificar como influenciam o modelo.
4. Explicar por que Cassandra não utiliza JOIN e como isso influencia a modelagem.

4. Estudo de Caso 3 — Modelagem de Dados em Redis

Tema: Sistema de Gerenciamento de Sessões e Carrinho de Compras em um E-commerce

🧭 Contexto do Problema

Uma plataforma de e-commerce precisa gerenciar:

1. Sessões de usuários em tempo real, com expiração automática.
2. Carrinhos de compras, permitindo adicionar, atualizar e remover produtos rapidamente.
3. Rankings de produtos mais vendidos ou mais acessados, de forma eficiente.
4. Histórico de atividades do usuário para recomendações e análise de comportamento.

O sistema deve ser altamente performático e capaz de lidar com milhões de requisições simultâneas. Redis é ideal aqui devido à sua natureza **in-memory**, suportando operações rápidas e estruturas de dados avançadas como **strings, hashes, lists, sets, sorted sets** e **bitmaps**.

⚙️ Escolha das Estruturas de Dados Redis

Caso de Uso	Estrutura Redis Sugerida	Justificativa
Sessão do usuário	**String** ou **Hash**	Permite expiração automática (TTL) e armazenamento de dados do usuário de forma simples.
Carrinho de compras	**Hash** ou **List**	Hashes permitem associar produtos e quantidades; lists podem manter ordem de adição.
Ranking de produtos	**Sorted Set (ZSet)**	Mantém produtos ordenados por número de vendas ou acessos.
Histórico de atividades	**List** ou **Stream**	Permite registrar ações ordenadas com limite de tamanho (`LPUSH + LTRIM`).

💡 Modelagem de Dados

1️⃣ Sessões de Usuário

Cada sessão é armazenada como uma chave única:

```
Key: session:<session_id>
Type: Hash
Fields:
user_id -> 12345
login_time -> 2025-10-12T22:00:00
last_activity -> 2025-10-12T22:30:00
```

```
TTL: 30 minutos
```

Operações comuns:

```
```bash
HSET session:abc123 user_id 12345 login_time 2025-10-12T22:00:00
EXPIRE session:abc123 1800  # Expira em 30 minutos
HGETALL session:abc123
```

2️⃣ Carrinho de Compras

Cada carrinho é armazenado como um hash:

```
Key: cart:<user_id>
Type: Hash
Fields:
    product:<product_id> -> quantity
    product:1001 -> 2
    product:1002 -> 1
```

Operações comuns:

```
HSET cart:12345 product:1001 2
HINCRBY cart:12345 product:1002 1
HDEL cart:12345 product:1001
HGETALL cart:12345
```

3️⃣ Ranking de Produtos

Ranking de produtos mais vendidos usando Sorted Set:
```

```
Key: product:sales
Type: Sorted Set
Members:
    product_id -> score (quantidade vendida)
```

Operações comuns:

```
ZINCRBY product:sales 1 1001   # Incrementa vendas do produto 1001
ZRANGE product:sales 0 9 WITHSCORES  # Top 10 produtos mais vendidos
```

4️⃣ Histórico de Atividades do Usuário

Mantido como List com tamanho máximo:

```
Key: history:<user_id>
Type: List
Values (LPUSH newest first):
    "viewed_product:1001"
    "added_to_cart:1002"
    "checkout"
```

Operações comuns:

```
LPUSH history:12345 "viewed_product:1001"
LTRIM history:12345 0 49  # Mantém apenas os últimos 50 eventos
LRANGE history:12345 0 -1  # Mostra todo o histórico
```

📊 Vantagens da Modelagem Redis

Alta Performance: Tudo em memória, ideal para acesso rápido.

Estruturas Especializadas: Sorted sets, hashes, lists permitem modelar dados complexos de forma natural.

Escalabilidade: Pode ser replicado e particionado (sharding) facilmente.

TTL e Expiração: Útil para sessões temporárias e caches.

Operações Atômicas: Comandos como HINCRBY e ZINCRBY garantem consistência sem transações complexas.

Tarefas (Responder)

1. Justificar Redis para cache e sessão.
2. Modelar um carrinho com 3 produtos usando comandos Redis.
3. Explicar a diferença entre List e Stream.
4. Criar uma solução para armazenar e consultar os produtos mais visualizados na última hora.

---

5. Estudo de Caso 4 — Modelagem de Dados em Neo4j

Tema: Sistema de Recomendação de Filmes

🧭 Contexto do Problema

Uma plataforma de streaming deseja implementar um sistema de recomendação de filmes. O objetivo é sugerir aos usuários filmes baseados em suas preferências, histórico de avaliações e afinidades com outros usuários.

O Neo4j é ideal para este cenário porque os relacionamentos entre usuários, filmes, gêneros e avaliações podem ser modelados naturalmente como grafos.

📊 Modelagem do Grafo

O grafo possui os seguintes **nós**:

- **Usuário (`User`)**
  Propriedades: `id`, `nome`, `idade`, `email`
- **Filme (`Movie`)**
  Propriedades: `id`, `titulo`, `ano`, `genero`
- **Gênero (`Genre`)**
  Propriedades: `nome`

E os **relacionamentos**:

- **(:User)-[:ASSISTIU]->(:Movie)**
  Propriedades: `nota`, `data`
- **(:Movie)-[:PERTENCE_A]->(:Genre)**

- **(:User)-[:AMIGO_DE]->(:User)**

  Para modelar conexões sociais e similaridade de gosto

🔍 Objetivos do Estudo de Caso

1. Inserir dados de usuários, filmes e gêneros no Neo4j.
2. Criar relacionamentos entre usuários, filmes e gêneros.
3. Consultar filmes assistidos por um usuário específico.
4. Recomendar filmes a um usuário com base no que seus amigos assistiram.
5. Identificar usuários com gostos similares.

🛠️ Estrutura de Dados de Exemplo

Usuários

id	nome	idade	email
1	Alice	25	alice@email.com
2	Bob	30	bob@email.com
3	Carol	22	carol@email.com

Filmes

id	titulo	ano	genero
1	Matrix	1999	Sci-Fi
2	Titanic	1997	Romance

id	titulo	ano	genero
3	O Senhor dos Anéis	2001	Fantasia

Gêneros

- Sci-Fi
- Romance
- Fantasia

Relacionamentos

- Alice assistiu Matrix (nota 5)
- Bob assistiu Titanic (nota 4)
- Alice é amiga de Bob

✅ Consultas Exemplares

1. Filmes assistidos por Alice:

```
MATCH (u:User {nome: "Alice"})-[:ASSISTIU]->(m:Movie)
RETURN m.titulo, m.ano
```

Tarefas (Responder)

1. Criar diagrama ASCII dos nós e relacionamentos.
2. Criar consulta Cypher recomendando filmes assistidos pelos amigos do usuário.
3. Explicar por que Neo4j é superior para consultas altamente relacionais.
4. Propor um novo relacionamento que melhore o sistema.