Modelagem de Banco de Dados NoSQL

Orientação a Acesso e Denormalização Estratégica

> **Se o SQL é tão bom, por que Google, Amazon e Netflix usam outras formas de banco de dados?**

Abertura: Por que NoSQL?

Se o SQL é tão bom, por que o Google, Amazon e Netflix usam outras formas de banco de dados?

Embora o modelo relacional (SQL) seja **excelente para integridade de dados**, ele pode ser **lento e inflexível** para volumes massivos de dados, estruturas complexas e acessos rápidos e específicos. O **NoSQL (Not Only SQL)** é a resposta para essa escala.

Volume Massivo de Dados

SQL sofre com JOINs em grandes volumes. NoSQL escala horizontalmente sem perda de performance.

Flexibilidade de Estrutura

SQL exige schema rígido. NoSQL permite estruturas dinâmicas que evoluem com a aplicação.

Velocidade de Acesso

NoSQL prioriza leitura rápida através de denormalização estratégica e orientação a acesso.

Desafios do Modelo Relacional em Escala

Lentidão de JOINs

PROBLEMA

Consultar dados de 5 tabelas relacionadas requer 4 JOINs. Com milhões de registros, cada JOIN multiplica o tempo.

IMPACTO

Consulta que deveria levar 100ms leva 5 segundos. Usuários esperam. Servidor sobrecarregado.

Rigidez de Schema

PROBLEMA

Adicionar um novo campo requer ALTER TABLE. Isso pode travar a tabela por horas em produção.

IMPACTO

Mudanças de negócio são lentas. Inovação é bloqueada. Competidores avançam mais rápido.

Escalabilidade Vertical

PROBLEMA

SQL escala verticalmente (servidor maior). Há limite: não existe servidor infinito. Custo exponencial.

IMPACTO

Custo operacional cresce rapidamente. Limite de capacidade é atingido. Ponto de falha único.

A Solução NoSQL: Orientação a Acesso

Orientação a Acesso é o oposto da normalização. No NoSQL, você estrutura os dados **da forma que eles serão consultados**, priorizando a **velocidade de leitura** em detrimento da redundância (que é aceitável na era do storage barato).

SQL: Normalização

Abordagem:

Eliminar redundância. Dados espalhados em múltiplas tabelas. Requer JOINs para consultar.

Vantagem:

Integridade de dados. Atualização em um lugar.

Desvantagem:

Lento com grandes volumes. Rígido. Escalabilidade limitada.

NoSQL: Denormalização

Abordagem:

Agrupar dados que são consultados juntos. Redundância aceitável. Uma consulta = um documento.

Vantagem:

Rápido. Flexível. Escalável horizontalmente.

Desvantagem:

Redundância. Atualizar múltiplos documentos é complexo.

Exemplo: E-commerce - Pedido com Itens

SQL (3 JOINs necessários):

```
SELECT p.*, c.*, i.*
FROM Pedidos p
JOIN Clientes c ON p.cliente_id = c.id
JOIN Itens i ON p.id = i.pedido_id
```

NoSQL (1 consulta simples):

```
db.pedidos.findOne({_id: "ped123"})
// Retorna cliente + itens aninhados
```

Tipos de Bancos de Dados NoSQL

Chave-Valor

EXEMPLOS

- Redis
- Memcached
- DynamoDB (AWS)

CARACTERÍSTICAS

Mais rápido e simples. Armazena pares chave-valor. Sem schema.

CASOS DE USO

Cache, sessões de usuário, contadores, leaderboards.

Documento

EXEMPLOS

- MongoDB
- CouchDB
- Firebase

CARACTERÍSTICAS

Mais popular. Armazena JSON/BSON. Permite estruturas aninhadas.

CASOS DE USO

E-commerce, CMS, aplicações web, catálogos de produtos.

Grafos

EXEMPLOS

- Neo4j
- Amazon Neptune
- ArangoDB

CARACTERÍSTICAS

Relacionamentos complexos. Nós e arestas. Queries rápidas.

CASOS DE USO

Redes sociais, recomendações, análise de fraude, conhecimento.

Modelagem Orientada a Documento: Embedded vs Reference

Embedded Data (Dados Aninhados)

Inserir o documento filho **dentro** do documento pai. Prioriza **leitura rápida**.

</> Exemplo JSON

```
{ "_id": "pedido_001", "cliente": { "nome": "João", "email": "joao@email.com" },
"itens": [ { "produto": "Notebook", "preco": 3500 }, { "produto": "Mouse",
"preco": 150 } ] }
```

QUANDO USAR

Dados sempre consultados juntos e que não mudam frequentemente (ex: itens de um pedido).

- ✓ Uma consulta retorna tudo
- ✓ Sem JOINs, muito rápido
- ✗ Redundância de dados
- ✗ Atualizar múltiplos docs

Reference Data (Dados Referenciados)

Armazenar apenas o **ID** do documento filho. Prioriza **integridade**.

</> Exemplo JSON

```
{ "_id": "pedido_001", "cliente_id": "cliente_123", "itens_ids": [ "item_001",
"item_002" ] } // Documento separado { "_id": "cliente_123", "nome": "João",
"email": "joao@email.com" }
```

QUANDO USAR

Dados que mudam frequentemente ou são muito grandes (ex: dados de cliente que podem ser atualizados).

- ✓ Sem redundância
- ✓ Atualizar uma vez
- ✗ Requer múltiplas consultas
- ✗ Mais lento (como JOINs)

Atividade 1: Análise de Mini-Casos

Instruções: Divida a turma em grupos de 3-4 alunos. Cada grupo escolhe um mini-caso abaixo e identifica as principais consultas que os usuários farão. Foco: O que o usuário vai buscar?

E-commerce

Contexto: Armazenar carrinho de compras com itens, sessão do usuário e dados básicos do cliente.

Principais Consultas:

- → Mostrar todos os itens do carrinho
- → Atualizar quantidade de um item
- → Remover item do carrinho
- → Calcular total do carrinho

Sua Tarefa:

Liste TODAS as consultas que um usuário faria ao usar o carrinho. Pense em casos especiais.

Clínica

Contexto: Prontuário eletrônico com dados do paciente, histórico de consultas e resultados de exames.

Principais Consultas:

- → Buscar prontuário completo de um paciente
- → Listar todas as consultas de um paciente
- → Buscar resultados de exames recentes
- → Adicionar nova consulta ao histórico

Sua Tarefa:

Liste TODAS as consultas que um médico ou recepcionista faria. Considere urgências.

Rede Social

Contexto: Armazenar posts com comentários aninhados, likes e informações do autor.

Principais Consultas:

- → Mostrar post com todos os comentários
- → Adicionar comentário a um post
- → Listar posts de um usuário
- → Contar likes de um post

Sua Tarefa:

Liste TODAS as consultas que um usuário faria ao navegar na rede social. Pense em feed.

Atividade 2: Definição de Coleções e Estrutura de Documentos

Instruções: Os grupos devem projetar o modelo NoSQL focando no modelo de documento (JSON) e no MongoDB ou DynamoDB como referência. Use papel, Miro ou Draw.io para visualizar.

1 Definir Coleções

Identifique quais serão as **"tabelas"** do seu modelo. Exemplo: **carrinhos**, **pacientes**, **posts**.

2 Estrutura do Documento

Desenhe a estrutura JSON de um documento completo. Inclua todos os campos que serão armazenados.

3 Decisão de Denormalização

Decida o que será **aninhado (Embedded)** e o que será **referenciado (apenas ID)**.

4 Definir Chaves

Defina a chave primária (**_id** no MongoDB) e as chaves de busca que serão usadas nas consultas.

5 Validar Consultas

Escreva as principais consultas. Se exigem muitos JOINs, a denormalização está bem feita.

6 Documentar Decisões

Registre **por que** cada decisão foi tomada. Exemplo: "Aninhamos itens porque são consultados juntos".

Exemplo de Estrutura JSON (E-commerce - Carrinho):

```
{ "_id": "carrinho_user_12345", "usuario_id": "user_12345", "usuario_nome": "João Silva", "itens": [ {"produto_id": "prod_001", "nome": "Notebook", "preco": 3500, "qtd": 1} ], "total": 3500, "status": "ativo" }
```

Validação Cruzada: Teste de Denormalização

Como Funciona

- Grupos **trocam os modelos** entre si
- Grupo revisor recebe a estrutura JSON do colega
- Tenta **escrever uma consulta SQL** para aquela estrutura
- Se o SQL exigir **muitos JOINs**, a denormalização está bem feita!

Critério de Sucesso

- **Denormalização bem feita** = SQL exigiria muitos JOINs
- Significa que você **agrupou bem** os dados que são consultados juntos
- Uma consulta NoSQL retorna tudo sem JOINs
- Performance em NoSQL será **muito superior** ao SQL equivalente

Exemplo: Validação Cruzada de Prontuário (Clínica)

Estrutura NoSQL (MongoDB):

```
{ "_id": "pac_123", "nome": "Maria", "consultas": [ { "data": "2025-01-10",
"medico": "Dr. João", "diagnostico": "..." } ], "exames": [...] }
```

VALIDAÇÃO: Uma consulta retorna tudo!

db.pacientes.findOne({_id: "pac_123"})

SQL Equivalente (3+ JOINs):

```
SELECT p.*, c.*, e.* FROM Pacientes p JOIN Consultas c ON p.id = c.pac_id JOIN
Exames e ON p.id = e.pac_id WHERE p.id = 'pac_123'
```

CONCLUSÃO: Denormalização bem feita!

O SQL exigiu 2 JOINs. NoSQL faz em 1 consulta. Sucesso!

Discussão Crítica: Vantagens, Desvantagens e Modelos Híbridos

Desvantagens do NoSQL

Anomalia de Atualização

Se o preço do produto mudar, teríamos que atualizar **todos os documentos de Pedidos antigos** onde o preço está aninhado. Isso é complexo e propenso a inconsistências.

```
// Problema: Preço mudou de 3500 para 3200 // Precisamos atualizar TODOS os pedidos antigos db.pedidos.updateMany( {"itens.produto_id": "prod_001"}, {$set: {"itens.$.preco": 3200}} )
```

Redundância de Dados

Dados aninhados causam redundância. Armazenar nome do cliente em cada pedido multiplica o tamanho do banco de dados.

Falta de Integridade

NoSQL não tem constraints como SQL. Você é responsável por validar dados na aplicação.

Modelos Híbridos (SQL + NoSQL)

Tendência Atual

Usar **SQL para transações financeiras** (integridade crítica) e **NoSQL para grandes catálogos e dados voláteis** (performance crítica).

Cenário	Escolha
Transações financeiras	**SQL** (integridade)
Catálogo de produtos	**NoSQL** (performance)
Carrinho de compras	**NoSQL** (velocidade)
Dados de usuário	**SQL** (consistência)
Feed de rede social	**NoSQL** (escala)
Relatórios analíticos	**SQL** (queries complexas)

Conclusão

A escolha do banco de dados deve ser orientada pela **demanda de acesso** e pelo **volume de dados**, e não por dogma.

Conclusão e Consolidação

Conceitos Principais Aprendidos

- **Orientação a Acesso:** Estruturar dados da forma que serão consultados, não como devem ser armazenados.
- **Denormalização Estratégica:** Redundância é aceitável quando melhora performance de leitura.
- **Embedded vs Reference:** Escolher entre aninhamento e referência baseado em frequência de acesso e mudança.
- **Escalabilidade Horizontal:** NoSQL escala distribuindo dados entre múltiplos servidores.
- **Flexibilidade de Schema:** Estrutura dinâmica permite evolução rápida da aplicação.

Checklist de Validação

- [ ] Identifiquei as principais consultas que os usuários farão.
- [ ] Defini as coleções (tabelas) do meu modelo NoSQL.
- [ ] Criei a estrutura JSON de cada documento.
- [ ] Decidi o que será Embedded e o que será Reference.
- [ ] Escrevi as principais consultas e validei que não precisam de muitos JOINs.
- [ ] Documentei as decisões de denormalização e justifiquei cada uma.

Reflexão Final: A Escolha Certa do Banco de Dados

A escolha entre **SQL e NoSQL** não é uma questão de dogma, mas de **demanda de acesso e volume de dados**. Use SQL para transações financeiras críticas e integridade garantida. Use NoSQL para grandes catálogos, dados voláteis e escalabilidade horizontal. Muitas empresas usam **ambos simultaneamente** (modelos híbridos). O importante é entender os trade-offs e escolher a ferramenta certa para cada problema.