**Bancos de Dados Não Relacionais**

**O que é um Banco de Dados Não Relacional?**

O termo correto para um banco de dados não relacional é NOT Only SQL, isto é, não apenas SQL, e sim mais abrangente. Eles não seguem o modelo de tabelas e relacionamentos. São projetados para lidar com alto volume de dados e possuem alta escalabilidade (em alguns casos). Além disso, possuem alta flexibilidade na estrutura de dados e são amplamente utilizados em cenários onde a consistência imediata dos dados não é crítica.

**Diferenças entre SQL e NoSQL:**

Tabela

O conteúdo gerado por IA pode estar incorreto.

**Tipos de Bancos de Dados NoSQL:**

Os bancos de dados não relacionais são divididos de acordo com os seus tipos de armazenamento.

* Key-Value (Chave-Valor);
* Documento;
* Coluna;
* Grafos;
* entre outros.

Key-Value > Chave-Valor:

Armazena dados como pares de chave e valor, onde cada chave é um identificador único para acessar seu valor correspondente. Exemplos de SGBDs: Redis, Riak, Amazon DynamoDB.

Esse tipo de BD possui características como uma escalabilidade horizontal muito boa (pode ser distribuído em vários servidores), tem um acesso muito rápido à informação e tem um sistema de armazenamento simples.

**Exemplo de Uso:** Armazenar informações de sessão do usuário.

Baseado em Documentos:

Armazenam dados em documentos semiestruturados, geralmente em formato JSON ou BSON. Exemplos de SGBDs: MongoDB, Couchbase, Apache CouchDB.

Algumas características desse BD são: flexibilidade de schema, modelagem de dados complexas (pois apesar de não ter um schema fixo, ele aceita uma grande variedade de informações possíveis de armazenamento) e tem consultas eficientes.

**Exemplo de Uso:** Catálogo de e-commerce.

Baseado em Colunas:

Armazenam dados em formato de colunas, o que permite alta escalabilidade e eficiência em determinados tipos de consultas. Exemplos de SGBDs: Apache Cassandra, ScyllaDB, HBase.

Algumas de suas características são a sua escalabilidade horizontal boa, eficiência na recuperação de dados e possui esquema flexível.

**Exemplo de Uso:** Armazenar registros de logs de aplicações.

Baseado em Grafos:

Armazenam e consultam dados interconectados, onde os relacionamentos entre os dados são tão importantes quanto os próprios dados. Exemplos de SGBDs: Neo4j, Amazon Neptune, JanusGraph.

**Exemplo de Uso:** Redes sociais podem usá-los para armazenar os perfis dos usuários e suas conexões.

**MongoDB:**

O MongoDB é um banco de dados não relacional orientado a documentos. Ele lida muito bem com grandes volumes de dados, possui escalabilidade horizontal e uma modelagem flexível, não exigindo um esquema. Porém, permite que os documentos sejam armazenados em formato BSON (Binary JSON), proporcionando uma estrutura semiestruturada.

Vantagens:

* Flexibilidade na modelagem de dados.
* Escalabilidade horizontal para lidar com grandes volumes de dados.
* Consultas ricas e suporte para consultas complexas.
* Alta disponibilidade e tolerância a falhas.
* Comunidade ativa e recursos de suporte.

Desvantagens:

* Menor consistência imediata em comparação com bancos de dados relacionais.
* Consultas complexas podem exigir um maior conhecimento e planejamento adequado.
* Maior consumo de espaço de armazenamento em comparação com bancos de dados relacionais devido à flexibilidade dos documentos.

Aplicações:

* Aplicações web: onde a escalabilidade e flexibilidade são cruciais para lidar com volumes variáveis de dados.
* Análise de big data: análise de grandes volumes de dados não estruturados ou semiestruturados.
* Armazenamento de dados semiestruturados: permite a inserção de documentos com estruturas diferentes em uma mesma coleção.
* Casos de uso de geolocalização.

Estrutura de um Documento Simples:

{

\_id: ObjectId(“”),

“nome\_campo”: “valor\_campo”,

“nome\_campo\_2”: “valor\_campo\_2”,

...

}

Modelagem orientado por consultas:

A modelagem de dados no MongoDB deve ser orientada pelas consultas que serão realizadas com mais frequência.

Inner Documents:

No MongoDB , é comum **desnormalizar** os dados para evitar operações de junção (join) custosas. Isso significa que os dados relacionados podem ser armazenados juntos em um único documento, em vez de serem distribuídos em várias coleções.

Devemos usá-los quando:

* Os dados aninhados são específicos para o documento pai.
* Os dados aninhados são sempre acessados juntamente com o documento pai.
* A cardinalidade do relacionamento é um-para-muitos.

Não devemos utilizá-los quando os dados aninhados precisarem ser consultados e atualizados independentemente do documento pai, nesse caso é mais adequado utilizar coleções separadas.

Referências:

São formas de relacionar os documentos entre si.

Devemos usá-las quando:

* Os dados têm seu próprio significado e podem ser acessados independentemente do documento pai.
* Os dados têm uma cardinalidade mais alta (por exemplo, vários usuários podem ter várias reservas).

Não devemos usá-las se os dados aninhados precisarem ser consultados e atualizados independentemente do documento pai, nesse caso é mais adequado usar coleções separadas.

Operações no MongoDB:

Criando um Database:

* **use** {{nome\_banco}}

Enquanto o database não tiver uma collection ele não será apresentado na lista.

Criando uma Collection:

* db.createCollection(“nome\_collection”)

Inserindo Documentos:

* db.nome\_collection.insertOne({})
* db.nome\_collection.insertMany([{}])

Consultando Documentos:

* db.nome\_collection.find({})
* db.nome\_collection.findOne({})
* db.nome\_collection.findOneAndUpdate({ }, {})
* db.nome\_collection.findOneAndDelete({})

Atualizando Documentos:

* db.nome\_collection.updateOne({}, {$operador: {}})
* db.nome\_collection.updateMany({}, {$operador: {}})
* db.nome\_collection.replaceOne({})

Operadores de Update:

* $inc 🡪 incrementador
* $push 🡪 adiciona na lista
* $set 🡪 define o valor
* $unset
* $rename

Excluindo Documentos:

* db.nome\_collection.deleteOne({})
* db.nome\_collection.deleteMany({})

Operadores Comuns:

* “:” 🡪 operador de igualdade

Operadores Lógicos:

* $and
  + { $and: [{ nome: "Felipe" }, { idade: 21 }] }
    - Retornará todos os usuários de nome Felipe e idade 21 anos.
* $or
  + { $or: [{ nome: "Felipe" }, { nome: “João” }] }
    - Retornará todos os usuários de nome Felipe ou João.
* $not
  + { idade: { $not: { $gt: 25 }} }
    - Retornará todos os usuários de 25 anos ou menos.

Operadores de Comparação:

* $eq 🡪 ==
  + {nome: {$eq: "Felipe"}}
    - Retornará todos os usuários de nome Felipe.
* $ne 🡪 !=
  + {nome: {$ne: "Felipe"}}
    - Retornará todos os usuários que não tiverem nome Felipe.
* $gt 🡪 >
  + {idade: {$gt: 20}}
    - Retornará todos os usuários maiores de 20 anos.
* $gte 🡪 >=
  + {idade: {$gte: 20}}
    - Retornará todos os usuários de 20 anos ou mais.
* $lt 🡪 <
  + {idade: {$lt: 20}}
    - Retornará todos os usuários menores de 20 anos.
* $lte 🡪 <=
  + {idade: {$lte: 20}}
    - Retornará todos os usuários de 20 anos ou menos.
* $in 🡪 [] igualdade dentro de um array
  + {cidade: {$in: [“São Paulo”, “Belo Horizonte”]}}
    - Retornará todos os usuários da cidade de São Paulo ou Belo Horizonte.
* $nin 🡪 [] negação de igualdade dentro de um array
  + {cidade: {$in: [“São Paulo”, “Belo Horizonte”]}}
    - Retornará todos os usuários fora da cidade de São Paulo ou Belo Horizonte.

**Redis:**

O Redis é um sistema de armazenamento de dados em memória de alto desempenho que é baseado no sistema de chave-valor. É muito utilizado em conjunto com outros bancos de dados.

Suas principais características são: armazenamento em memória, estrutura de dados versátil, operações atômicas, cache de alto desempenho e Pub/Sub (Publicação/Assinatura).

Principais Comandos:

* SET 🡪 adicionar informação no Redis
  + SET chave “Valor”
* GET 🡪 retornar informação no Redis
  + GET chave
* DEL 🡪 deleta uma informação do Redis
  + DEL chave
* EXISTS 🡪 verifica se existe uma chave dentro do Redis
  + EXISTS chave
* KEYS 🡪 lista todas as chaves correspondente ao padrão (pattern) utilizado
  + KEYS pattern
    - KEYS \*name\* 🡪 retorna todas as chaves que tenham “name” em seu nome (ex: firstname, lastname)
    - KEYS \* 🡪 retorna todas as chaves
* INCR 🡪 incrementa uma variável numérica
  + INCR chave 🡪 observação: esse comando incrementará o valor associado à chave
* DECR 🡪 decrementa uma variável numérica
  + DECR chave