



Bancos de Dados NoSql

Fabrício Tonetto Londero

Contextualização

- Ao longo das últimas décadas, os bancos de dados relacionais dominaram o desenvolvimento de sistemas, com estruturas rigidamente organizadas em tabelas, chaves primárias e estrangeiras, além de uso extensivo da linguagem SQL.
- No entanto, com o crescimento de sistemas web, redes sociais, dispositivos móveis e a chegada do Big Data, surgiram novas demandas: mais velocidade, mais flexibilidade e maior capacidade de escalar horizontalmente.



Contextualização

-
- Nesse cenário, emergem os bancos de dados NoSQL, cuja sigla significa “*Not Only SQL*”.
 - Eles representam uma nova categoria de bancos que não utilizam exclusivamente o modelo relacional e nem a linguagem SQL tradicional.
 - São sistemas criados para lidar com grandes volumes de dados, alta taxa de transações, dados não estruturados ou semiestruturados e com necessidade de alta disponibilidade e escalabilidade.



Classificação

- 1. Chave/Valor
- 2. Orientado a Documentos
- 3. Colunar (Wide-Column)
- 4. Orientado a Grafos

1. Chave/Valor



Os bancos de dados do tipo **chave/valor** são a forma mais simples de armazenamento NoSQL.



Eles funcionam como um grande dicionário ou mapa, onde cada valor é associado a uma chave única.



A chave atua como identificador único do dado, permitindo acesso direto, sem a necessidade de varreduras em tabelas.



O valor pode ser uma informação simples, como uma *string* ou número, ou até mesmo estruturas mais complexas, como objetos JSON, listas ou *hashes*.



Como não há esquema definido, o desenvolvedor tem total liberdade para definir o conteúdo armazenado.

1. Chave/Valor

Principais usos:

- Sistemas de cache, como o armazenamento temporário de páginas visitadas frequentemente.
- Gerenciamento de sessões de usuários, armazenando dados como preferências, idioma e status de login.
- Ranking de jogadores em jogos online, contadores, notificações e configurações de sistemas.

Vantagens:

- Simplicidade na estrutura e operação.
- Alta performance para gravação e leitura.
- Excelente escalabilidade horizontal.

Desvantagens:

- Limitações em buscas por campos internos, já que só a chave é indexada.
- Não há suporte a relacionamentos complexos entre dados.

Exemplos de bancos chave/valor:

- Redis
- Amazon DynamoDB
- Riak KV

1. Chave/Valor

Imagine uma agenda telefônica:

```
{  
  "usuario_101":  
    {  
      "nome": "Fabrício",  
      "email": "fabricio@email.com",  
      "idade": 35,  
      "cidade": "Porto Alegre"  
    }  
}
```

- No exemplo, usuario_101 é a chave e o valor é um objeto JSON com as propriedades nome, email, idade e cidade.
- Nesse formato, não há necessidade de tabelas, colunas ou esquemas rígidos. O desenvolvedor simplesmente define o que será armazenado no valor, e o banco cuida da indexação da chave.

1. Chave/Valor

- Apesar da simplicidade, o modelo chave/valor exige atenção quanto à estrutura dos valores.
- Como não há relações entre os dados, buscas mais complexas podem ser difíceis ou custosas.
- Por isso, **é comum usá-lo em conjunto com outros modelos**, dentro de arquiteturas de microserviços ou sistemas modulares.

1. Chave/Valor

- `SET usuario:001 '{"nome": "Fabrício", "idade": 34, "cidade": "Porto Alegre"}'`
- Exemplo no banco redis.io.
- O comando acima, armazena um novo usuário, com a chave usuário:001



The screenshot shows the Redis command-line interface (CLI) with a dark background. At the top, a command is entered: `1 SET usuario:001 '{"nome": "Fabrício", "idade": 34, "cidade": "Porto Alegre"}'`. Below the command, the response `2` is shown. At the bottom of the interface, there is a status bar that displays the command and response: `SET usuario:001 '{"nome": "Fabrício", "idade": 34, "cidade": "Porto Alegre"}'` followed by `"OK"`. Above the status bar, there are navigation buttons labeled "Tutorials:", "Intro to search", "Basic use cases", and "Intro to vector search".

1. Chave/Valor

- SET usuario:001 '{"nome": "Ana", "idade": 22, "cidade": "São Paulo"}'
- SET usuario:002 '{"nome": "Bruno", "idade": 28, "cidade": "Rio de Janeiro"}'
- SET usuario:003 '{"nome": "Carla", "idade": 35, "cidade": "Belo Horizonte"}'
- SET usuario:004 '{"nome": "Daniel", "idade": 19, "cidade": "Salvador"}'
- SET usuario:005 '{"nome": "Elaine", "idade": 42, "cidade": "Fortaleza"}'
- SET usuario:006 '{"nome": "Fernando", "idade": 31, "cidade": "Porto Alegre"}'
- SET usuario:007 '{"nome": "Gabriela", "idade": 27, "cidade": "Curitiba"}'
- SET usuario:008 '{"nome": "Henrique", "idade": 24, "cidade": "Manaus"}'
- SET usuario:009 '{"nome": "Isabela", "idade": 30, "cidade": "Natal"}'
- SET usuario:010 '{"nome": "João", "idade": 33, "cidade": "Recife"}'

1. Chave/Valor

- Exemplos
 - GET usuario:001
 - KEYS usuario:*
 - DEL usuario:004
- Para poder fazer uma consulta com filtro, devemos indexar o conteúdo do filtro do JSON, pois a ferramenta não consegue acessar diretamente
 - SET nome: Gabriela usuario:007
 - GET nome: Gabriela

```
1 KEYS usuario:*
2
```

Tutorials: [Intro to search](#) [Basic u](#)

KEYS usuario:*

- 1) "usuario:006"
- 2) "usuario:003"
- 3) "usuario:009"
- 4) "usuario:004"
- 5) "usuario:001"
- 6) "usuario:005"
- 7) "usuario:002"
- 8) "usuario:008"
- 9) "usuario:010"
- 10) "usuario:007"

```
1 GET usuario:001
```

Tutorials: [Intro to search](#) [Basic use cases](#) [Intro to vector search](#)

GET usuario:001

```
"{"nome": "Fabr\x3\xadcio", "idade": 34, "cidade": "Porto Alegre"}"
```

2. Orientado a Documentos



Diferente do modelo chave/valor, os bancos orientados a documentos armazenam os dados em forma de documentos independentes, normalmente no formato JSON, BSON ou XML.



Cada documento contém os dados e a estrutura (chaves e valores) de forma autônoma, o que traz grande flexibilidade.



Esse modelo é ideal para aplicações onde os dados possuem estruturas variadas ou aninhadas, como perfis de usuários, catálogos de produtos e conteúdos dinâmicos.



2. Orientado a Documentos

- **Cada documento pode ter campos diferentes, sem a rigidez de esquemas fixos.**
- Um dos grandes diferenciais é que, ao contrário dos bancos chave/valor, aqui é possível realizar consultas por qualquer campo do documento, aplicar filtros, ordenações e índices secundários com eficiência.

2. Orientado a Documentos

Principais usos:

- Aplicações web modernas, especialmente em conjunto com **APIs RESTful**.
- Sistemas de e-commerce, onde cada produto pode ter atributos diferentes.
- Armazenamento de conteúdo dinâmico, como blogs, fóruns, catálogos, etc.

Vantagens:

- Flexibilidade no formato dos dados.
- Boa performance para leitura e escrita.
- Suporte a consultas complexas por campos internos.
- Fácil integração com aplicações *frontend* que usam JSON.

Desvantagens:

- Pode consumir mais espaço em disco devido à estrutura dos documentos.
- Atualizações em massa são mais complexas.

Exemplos de bancos orientados a documentos:

- MongoDB
- CouchDB
- ArangoDB

2. Orientado a Documentos

- Buscar comentários feitos depois de 01/01/2000


Filter  { "name": "Mercedes Tyler" }

QUERY RESULTS: 1-20 OF MANY

```
_id: ObjectId('5a9427648b0beeb69579e7')
name: "Mercedes Tyler"
email: "mercedes_tyler@fakegmail.com"
movie_id: ObjectId('573a1390f29313caabcd4323')
text: "Eius veritatis vero facilis quaerat fuga temporibus. Praesentium exped..."
date: 2002-08-18T04:56:07.000+00:00
```



```
_id: ObjectId('5a9427648b0beeb6958131')
name: "Mercedes Tyler"
email: "mercedes_tyler@fakegmail.com"
movie_id: ObjectId('573a1392f29313caabcd8ac')
text: "Dolores nulla laborum doloribus tempore harum officiis. Rerum blanditi..."
date: 2007-09-21T08:52:00.000+00:00
```

Filter  { "date": { "\$gte": ISODate("2000-01-01T00:00:00Z") } }

QUERY RESULTS: 1-20 OF MANY

```
_id: ObjectId('5a9427648b0beeb69579e7')
name: "Mercedes Tyler"
email: "mercedes_tyler@fakegmail.com"
movie_id: ObjectId('573a1390f29313caabcd4323')
text: "Eius veritatis vero facilis quaerat fuga temporibus. Praesentium exped..."
date: 2002-08-18T04:56:07.000+00:00
```



```
_id: ObjectId('5a9427648b0beeb6957a38')
name: "Yara Greyjoy"
email: "gemma_whelan@gameofthron.es"
movie_id: ObjectId('573a1390f29313caabcd587d')
text: "Nobis incidunt ea tempore cupiditate sint. Itaque beatae hic ut quis..."
date: 2012-11-26T11:00:57.000+00:00
```

2. Orientado a Documentos

"\$options": "i" = case-insensitive
(ignora maiúsculas/minúsculas)

Filter 

```
{ "text": { "$regex": "dolor", "$options": "i" } }
```

QUERY RESULTS: 1-20 OF MANY

```
_id: ObjectId('5a9427648b0beeb69579f5')
name: "John Bishop"
email: "john_bishop@fakegmail.com"
movie_id: ObjectId('573a1390f29313caabcd446f')
text: "Id error ab at molestias dolorum incidunt. Non deserunt praesentium do..."
date: 1975-01-21T00:31:22.000+00:00
```

```
_id: ObjectId('5a9427648b0beeb6957a4b')
name: "Gregor Clegane"
email: "hafthor_julius_bjornsson@gameofthron.es"
movie_id: ObjectId('573a1390f29313caabcd5b9a')
text: "Voluptatum voluptatem nam et accusamus ullam qui explicabo exercitatio..."
date: 2015-02-08T01:28:23.000+00:00
```



2. Orientado a Documentos

- Mostra quantos comentários um usuário fez

sample_mflix.comments

STORAGE SIZE: 7.48MB LOGICAL DATA SIZE: 11.14MB TOTAL DOCUMENTS: 41079 INDEXES TOTAL SIZE: 2.12MB

Find Indexes Schema Anti-Patterns 0 **Aggregation** Search Indexes

Pipeline  \$match \$count

[+ CREATE NEW](#) [EXPORT TO LANGUAGE](#)

```
1 [
2   { "$match": { "name": "John Bishop" } },
3   { "$count": "comentarios_do_usuario" }
4 ]
```

PIPELINE OUTPUT
Sample of 1 document

comentarios_do_usuario : 237

2. Orientado a Documentos

- Retorna os 5 usuários que mais comentaram



The screenshot displays a MongoDB query interface. On the left, a code editor shows a MongoDB aggregation pipeline. On the right, the 'PIPELINE OUTPUT' section shows a sample of 5 documents, with a scrollable list of results.

```
1 [
2   { "$group": { "_id": "$name", "total": { "$sum": 1 } },
3   { "$sort": { "total": -1 } },
4   { "$limit": 5 }
5 ]
6 |
```

PIPELINE OUTPUT
Sample of 5 documents

OUTPUT OPTIONS

- `_id: "Mace Tyrell"`
`total: 277`
- `total: 260`
`_id: "Rodrik Cassel"`
- `_id: "The High Sparrow"`
`total: 260`
- `total: 258`
`_id: "Miskandei"`

3. Colunar (Wide-Column)

- Os bancos colunares organizam os dados por colunas em vez de linhas. **Isso significa que os valores de uma mesma coluna são armazenados fisicamente juntos**, o que permite uma leitura muito mais eficiente quando se deseja acessar apenas algumas colunas específicas de grandes conjuntos de dados.
- Esse modelo é especialmente indicado para sistemas analíticos, *business intelligence* (BI) e *data warehouses*, nos quais é comum realizar consultas agregadas sobre milhões de registros, mas envolvendo poucas colunas.
- Embora bancos relacionais também possuam colunas, os bancos colunares armazenam os dados de maneira fundamentalmente diferente, otimizando consultas analíticas.

3. Colunar (Wide-Column)

Principais usos:

- Processamento de grandes volumes de dados históricos.
- Aplicações de análise preditiva, mineração de dados e *dashboards*.
- Armazenamento de logs e métricas de sistemas em tempo real.

Vantagens:

- Alta performance em consultas analíticas que envolvem grandes volumes de dados.
- Boa compressão de dados.
- Suporte a escalabilidade horizontal.

Desvantagens:

- Não são indicados para operações transacionais pequenas e frequentes.
- Modelo mais complexo para iniciantes.

Exemplos de bancos colunares:

- Apache Cassandra
- Hbase
- Google Bigtable

3. Colunar (Wide-Column)

Aspecto	Relacional (SQL)	Colunar (Cassandra)
Linguagem	SQL	CQL (parecida com SQL)
Esquema fixo	Sim (tabelas pré-definidas)	Sim, mas mais flexível
Normalização	Comum (evita repetição)	Desnormalização (repete dados para performance)
Relacionamentos	Com <u>JOINS</u>	Não suporta <u>JOINS</u>

3. Colunar (Wide-Column)

- Comando CQL

```
token@cqlsh> use default_keyspace;
token@cqlsh:default_keyspace> create table vendas (produto_id text, data date, quantidade int, valor decimal, primary key(produto_id, data));
token@cqlsh:default_keyspace> insert into vendas (produto_id, data, quantidade, valor) values ('P001', '2025-05-05', 2, 59.90);
token@cqlsh:default_keyspace> insert into vendas (produto_id, data, quantidade, valor) values ('P002', '2025-05-08', 3, 89.70);
token@cqlsh:default_keyspace> insert into vendas (produto_id, data, quantidade, valor) values ('P003', '2025-05-22', 1, 120);
token@cqlsh:default_keyspace>
```

```
token@cqlsh:default_keyspace> select * from vendas
... ;
```

produto_id	data	quantidade	valor
P003	2025-05-22	1	120
P001	2025-05-05	2	59.90
P002	2025-05-08	3	89.70

(3 rows)

```
token@cqlsh:default_keyspace> select * from vendas where produto_id = 'P001'
... ;
```

produto_id	data	quantidade	valor
P001	2025-05-05	2	59.90

(1 rows)



3. Colunar (Wide-Column)

O CQL (Cassandra Query Language) foi propositalmente projetado para se assemelhar ao SQL, para tornar a curva de aprendizado mais amigável para quem já trabalha com bancos relacionais.

Mas, a sintaxe é parecida, mas a semântica, a arquitetura e os princípios de uso são bem diferentes.

4. Orientado a Grafos



Os bancos orientados a grafos são uma categoria especial de banco de dados projetada para armazenar e consultar dados altamente conectados.



Em vez de tabelas, documentos ou colunas, os dados são modelados como um grafo, composto por nós (vértices) e arestas (relacionamentos).



Cada nó representa uma entidade (por exemplo, uma pessoa ou produto), e cada aresta representa uma conexão entre essas entidades (por exemplo, amizade, compra, referência).



Esse modelo é especialmente eficiente para navegar em conexões complexas, pois foi projetado para que relacionamentos possam ser percorridos diretamente, sem a necessidade de *joins* custosos como em bancos relacionais.

4. Orientado a Grafos



Estrutura básica:

Nó (Node): representa uma entidade.

- Ex: um usuário, um produto, uma cidade.

Aresta (Edge): representa a relação entre dois nós.

- Ex: “é amigo de”, “mora em”, “comprou”.

Propriedades: tanto nós quanto arestas podem conter atributos (ex: nome, idade, data da relação, etc.).



Exemplo prático:

Imagine um sistema de rede social. Um grafo pode representar os usuários como nós e as amizades como arestas. Isso permite responder de forma extremamente eficiente a perguntas como:

“Quem são os amigos dos amigos do usuário X?”

“Qual o caminho mais curto entre dois usuários?”

4. Orientado a Grafos



Principais usos:

- **Redes sociais:** análise de conexões, influenciadores e comunidades.
- **Recomendações:** produtos comprados por usuários com perfis similares.
- **Detecção de fraudes:** relacionamentos suspeitos entre contas, transações e dispositivos.
- **Mapas e rotas:** navegação, logística e geolocalização.

Vantagens:

- Modelo de dados natural para aplicações com muitas interconexões.
- Consultas altamente performáticas em dados relacionais complexos.
- Flexível e visualmente intuitivo.

Desvantagens:

- Menor maturidade em comparação com outros modelos NoSQL.
- Não é o modelo ideal para dados tabulares ou transações de alta frequência simples.

Exemplos de bancos orientados a grafos:

- Neo4j - o mais popular e amplamente usado.
- OrientDB - híbrido com suporte a documentos e grafos.
- ArangoDB - também multimodelo, suporta grafos, documentos e chave/valor.
- Amazon Neptune - serviço gerenciado de grafo na nuvem.

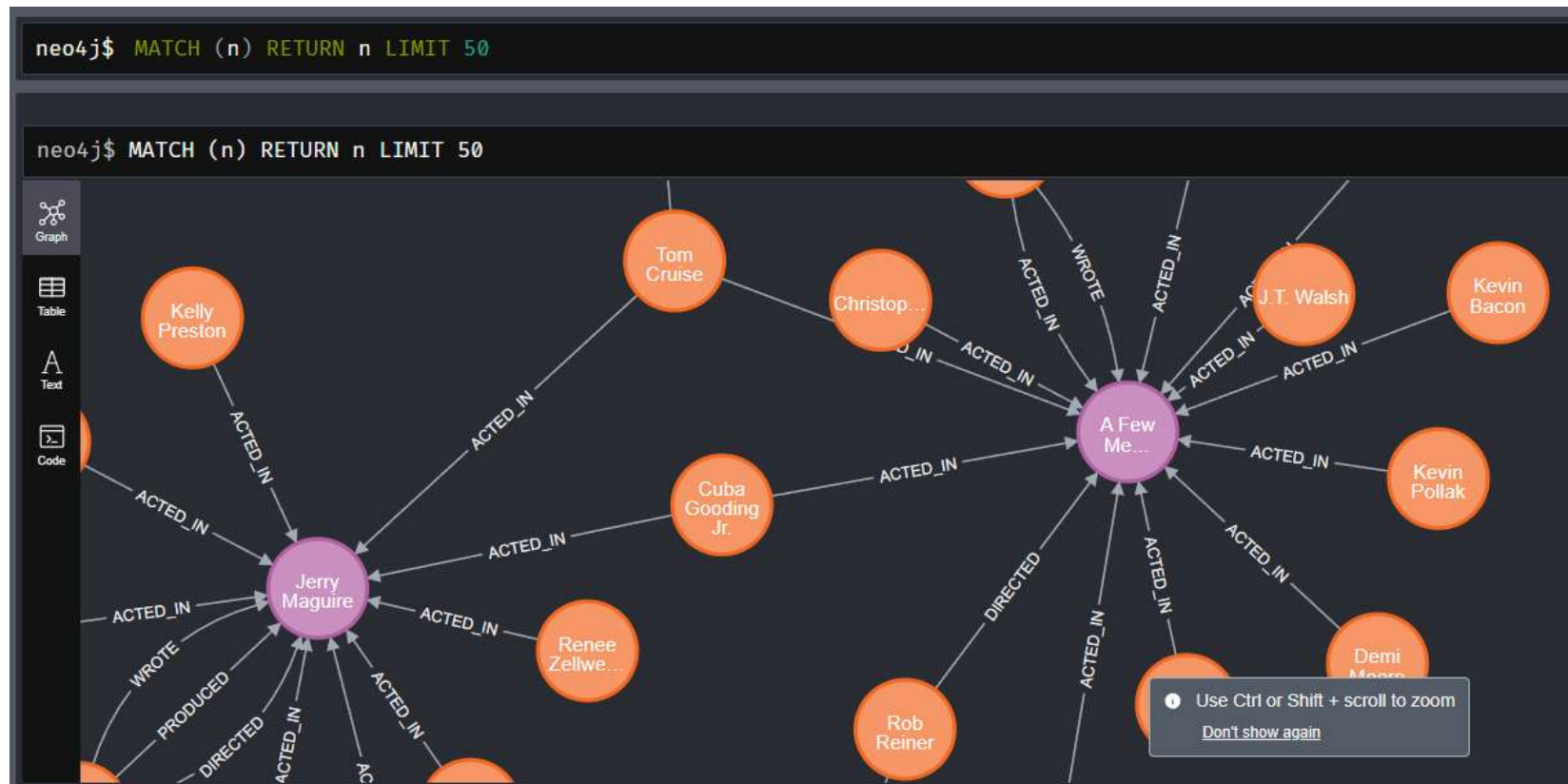
4. Orientado a Grafos

- Bancos de grafos normalmente utilizam linguagens próprias, como:
 - **Cypher** (Neo4j)
 - **Gremlin** (Apache TinkerPop)
 - **SPARQL** (em bancos RDF, como Virtuoso)

4. Orientado a Grafos

- Utilizando o Neo4j e seu banco modelo (Movies), com a linguagem Cypher
 - **MATCH (n) RETURN n LIMIT 50**
 - n = qualquer nó
 - RETURN n = retorna os nós
 - LIMIT 50 = retorna até 50 nós
- Exibe todos os nós e suas conexões no banco de dados

4. Orientado a Grafos



4. Orientado a Grafos

```
neo4j$ MATCH (m:Movie) RETURN m.title, m.released, m.tagline LIMIT 10
```

```
neo4j$ MATCH (m:Movie) RETURN m.title, m.released, m.tagline LIMIT 10
```

	m.title	m.released	m.tagline
1	"The Matrix"	1999	"Welcome to the Real World"
2	"The Matrix Reloaded"	2003	"Free your mind"
3	"The Matrix Revolutions"	2003	"Everything that has a beginning has an end"
4	"The Devil's Advocate"	1997	"Evil has its winning ways"
5	"A Few Good Men"	1992	"In the heart of the nation's capital, in a court"
6	"Top Gun"	1986	"I feel the need, the need for speed."

4. Orientado a Grafos

```
neo4j$ MATCH (a:Person) WHERE (a)-[:ACTED_IN]→() RETURN a.name LIMIT 10
```

	a.name
1	"Keanu Reeves"
2	"Carrie-Anne Moss"
3	"Laurence Fishburne"
4	"Hugo Weaving"
5	"Emil Eifrem"
6	"Charlize Theron"

```
neo4j$ MATCH (a:Person)-[:ACTED_IN]→(m:Movie {title: "The Matrix"}) RETURN a.name
```

	a.name
1	"Emil Eifrem"
2	"Hugo Weaving"
3	"Laurence Fishburne"
4	"Carrie-Anne Moss"
5	"Keanu Reeves"

4. Orientado a Grafos

