UM ESTUDO EXPLANATÓRIO ACERCA DOS MODELOS DE BANCO DE DADOS NOSQL

Alex Sandro Romeo de Souza POLETTO, Kaio Luiz BEGOSSO
apoletto@femanet.com.br, kaiobegosso@hotmail.com

RESUMO: NOSQL é uma tecnologia mundialmente em ascensão, e que deverá proporcionar soluções melhores e essenciais para empresas que necessitam de maior escalabilidade em seus bancos, necessidade de armazenamento de dados não estruturados e diminuição dos custos com banco de dados devido a maioria dos modelos NOSQL serem open source. O objetivo deste artigo é fazer um estudo acerca de alguns dos tipos de bancos de dados NoSQL disponíveis.

PALAVRAS-CHAVE: Bancos de dados; NoSQL; Open source; Soluções.

ABSTRACT: NOSQL is a technology world-wide rising and should provide better and essential solutions for companies that need greater scalability in their banks, unstructured data storage and decrease database costs because most NOSQL models are open source. The purpose of this article is to study some of the types of NoSQL databases available.

KEYWORDS: Data Base; Data; NoSQL; Open Source; Solutions

1. INTRODUÇÃO

Ao longo dos últimos anos a crescente demanda por bancos que suportem cada vez mais e mais dados fez com que os modelos de banco de dados tradicionais (relacionais) chegassem a um impasse, surge então a necessidade de novas alternativas, como os bancos de dados não relacionais NoSQL(Not Only SQL). Atualmente em empresas como Google, IBM, Yahoo, Facebook entre outras, o NoSQL já é utilizado em larga escala e aprimorado pelas mesmas, como por exemplo o Apollo do Facebook, entre os

modelos NoSQL que surgiram, destacam-se quatro modelos, que são, Chave-Valor, Documento, Colunas e Grafos.

Um dos grandes desafios atualmente na área de Computação é a manipulação e processamento de grande quantidade de dados no contexto de Big Data. O conceito Big Data pode ser resumidamente definido como uma coleção de bases de dados tão complexa e volumosa que se torna muito difícil (ou impossível) e complexa fazer algumas operações simples (e.g., remoção, ordenação, sumarização) de forma eficiente utilizando Sistemas Gerenciadores de Bases de Dados (SGBD) tradicionais. Por causa desse problema, e outros demais, um novo conjunto de plataformas de ferramentas voltadas para Big Data tem sido propostas, como por exemplo, Apache Hadoop[VIEIRA,FIGUEIREDO,et.al.,2012].

Uma das tendências para solucionar os diversos problemas e desafios gerados pelo contexto Big Data é o movimento denominado NoSQL (Not only SQL). NoSQL promove diversas soluções inovadoras de armazenamento e processamento de grande volume de dados.Estas soluções foram inicialmente criadas para solucionar problemas gerados por aplicações, por exemplo, Web 2.0 que na sua maioria necessitam operar com grande volume de dados, tenham uma arquitetura que “escale” com grande facilidade de forma horizontal, permitam fornecer mecanismos de inserção de novos dados de forma incremental e eficiente, além da necessidade de persistência dos dados em aplicações nas nuvens (*cloud computing*)[VIEIRA,FIGUEIREDO,et al.,2012].

2. TIPOS DE BANCOS NOSQL

Atualmente existem diversos tipos de modelos de Bancos de Dados NoSQL e cada um possui características distintas. Entre os que mais se destacam são: Chave-Valor, Documentos, Colunas e Grafos.

Chave-Valor: O modelo chave-valor é o mais simples e consiste essencialmente em uma grande tabela *hash* onde um valor é associado a uma chave única que é utilizada para recuperar os dados no BD. Voldemort e Riak são exemplos de BDs desta categoria[LIMA, MELO, 2015].

Grafos: Tipicamente, grafos podem ser definidos como uma abstração matemática que podem ser representados através de vértices e arestas, representando caminhos. No cenário atual de modelo de base de dados baseados em grafos, temos banco de dados relacionais e estruturas de web semântica. Em um banco de dados relacional é

necessário abstrair a normalização da modelagem ao nível máximo para representar um grafo, o que torna o tempo da consulta grande, devido ao número de consultas que será necessário ser feito. Isso acontece especialmente em estruturas recursivas de grafos, como percursos em largura ou em profundidade, pois é necessário guardar a referência a cada interação do percurso [TOTH].

Colunas: O modelo de dados família de colunas organiza seus dados em linhas e colunas e a sua grande característica encontra-se na sua abordagem desnormalizada para estruturar dados esparsos. BDs de família de colunas podem ser vistos como a exploração de dados tabulares com a divisão das colunas em grupos conhecidos como família-coluna. Cassandra e Hbase são exemplos de BDs de família de colunas [LIMA, MELO, 2015].

Documento: Um banco de dados baseado em documento é, na sua essência, parecido com a abordagem chave/valor com uma grande exceção. Em vez de armazenar qualquer arquivo binário como valor de uma chave é requisitado que o valor dos dados que serão armazenados possua um formato que o banco possa interpretar. Na maioria das vezes podem ser arquivos de extensão xml, json, json-blob ou qualquer formato descritivo, variando muito de ferramenta para ferramenta [TOTH].

3. CONCEITOS NECESSÁRIOS

Para melhor entendimento e com o objetivo de ter conhecimento sobre qual sistema escolher e como o mesmo vai operar, existem alguns conceitos necessários. Falaremos sobre os dois principais a seguir.

Teorema CAP – O teorema CAP (Consistency, Availability, Partition-tolerance) discute a relação entre consistência, disponibilidade e tolerância a partição.

Desta relação surgem três tipos de combinações:

CA – Sistema com consistência e disponibilidade alta, em contrapartida a tolerância a partição é baixa.

CP – Sistema com consistência e tolerância a partição alta, logo a disponibilidade é baixa.

AP – Sistema com disponibilidade e tolerância a partição alta, então a consistência tende a ser baixa.

Nos sistemas citados anteriormente, não significa que só se pode escolher dois entre os três tipos (Consistency, Availability, Partition-tolerance), mas sim de que pelo menos um a participação vai ser baixa. Sendo assim no ponto em que participação mais baixa será um ponto crítico.

ACID - ACID define um conjunto de princípios para bancos de dados orientados a transações, que é formado por:

Atomicidade (Atomicity) — Todo o conjunto de modificações deve ser tratado como uma unidade. Deve ser materializado, ou tudo, ou nada.

Consistência (Consistency) — Os dados devem se manter íntegros. Apenas dados válidos devem ser inseridos em uma transação.

Isolamento (Isolation) — As modificações temporárias de uma transação não devem refletir dentro de outras transações.

Durabilidade (Durability) — Após o final da transação, o banco de dados deve garantir que as alterações sobrevivam a falhas.

Lembrando que os dois teoremas fazem uso da palavra consistência, porem o significado em cada um é distinto. Sendo que no CAP consistência se refere à consistência atômica onde se refere que todas as operações são enfileiradas e sempre executadas como na ordem de uma FIFO. E no ACID consistência se refere à integridade, tanto referencial quanto a não violação das *constraints* no banco de dados.

4. COMPARATIVO ENTRE BANCOS

- **Cassandra**

Conforme dito no tópico anterior, a estrutura principal dos bancos colunares é basicamente uma grande tabela. De forma simplória, os bancos de colunas são os que mais se assemelham aos bancos relacionais por terem uma "tabela", mesmo que, na verdade eles, sejam muito diferentes. Os bancos de colunas surgiram com um trabalho do Google publicado sob nome “Bigtable: A Distributed Storage System for Structured Data” (GHEMAWAT; HSIEH; WALLACH; BURROWS et al.,2006).

Assumindo que o *keyspace* (banco de dados onde as tabelas existirão) esteja criado, para se criar uma nova tabela basta dar o seguinte comando:

CREATE TABLE NomeTabela(

id abcid PRIMARY KEY,
NOME_DA_COLUNA1 TIPO_DE_DADO_DA_COLUNA1,
NOME_DA_COLUNA2 TIPO_DE_DADO_DA_COLUNA2);

Podemos também passar o comando DESCRIBE TABLE onde todos os detalhes da tabela serão informados. Para inserir dados, usamos:

INSERT INTO nome_da_tabela (col1, col2, col3) VALUES (val1, val2, val3);
E para visualizar utilizamos:
SELECT colunas FROM nome_da_tabela;

Conforme pode se observar, os comandos são praticamente os mesmos ao SQL, e assim também é com os comandos UPDATE e DELETE vide exemplo:

UPDATE nome_da_tabela SET
atributo1='valor1',
atributo2='valor2'
WHERE id = abc123;
DELETE fromnome_da_tabela
WHERE id = abc123;

- **Riak**

Para criar um objeto no Riak, precisa-se de três valores: chave, valor e *bucket*. A maior parte dos bancos chave/valor usa apenas dois parâmetros, mas o Riak possibilita uma separação entre as chaves, que é chamado internamente de *bucket*. Desta forma, podemos usar o mesmo cluster de servidores para mais de uma finalidade, sem precisar ficar misturando as chaves entre as aplicações.

Para se criar um objeto no Riak, utiliza-se o seguinte comando: riak_object:new("*bucket*","chave","valor"). E um exemplo seria:

serie = riak_object new("temporada","episodiox","titulo").

Porém neste momento o objeto não está inserido no banco, para tal é necessário chamar a função *put*, desse modo: Conexao:put(serie, 1). Considerando que a conexão já esteja funcionando. Para selecionar este objeto é necessário invocar a seguinte função: riak_object:get_value(busca). Por exemplo:

{ok, busca}=Conexao:get(“temporada”,”episodiox”,1)

Riak_object:get_value(busca).

Agora nos falta atualizar e deletar para encerrarmos os comandos básicos de manipulação no riak. Então para atualizar os objetos no riak o comando necessário é:

{ok, Obj} = conexao:get(objId, {<<”temporada”>>,<<”episodiox”>>},<<”titulo”>>)

UpdateObj = riak_obj:update(Obj, <<”titulo1”>>)

Conexao:put(objId, UpdateObj, [return_body]).

E agora para deletar o comando que será necessário é:

{ok, Obj} = conexao:get(objId, {<<”temporada”>>,<<”episodiox”>>},

<<”titulo1”>>,[{deleted_vclock}]).

Com isso, encerramos os comandos básicos no Riak.

- **Neo4j**

Só existem dois tipos de dados no Neo4j, o nó e o relacionamento, e normalmente armazenamos nossas "entidades" nos nós. Cada nó individualmente pode ser comparado com um documento do MongoDB. Eles podem ter quantos e quais atributos desejarmos. Cada nó também pode ter um tipo. O comando para criar um nó é:

CREATE (nome_de_variavel: TipoDoNo {prop1: valor1, prop2: valor2}).

É necessário que se tenha dois nós para então definir o relacionamento entre eles. Para criar um relacionamento, dependemos de referências para os nós que serão unidos. A maneira mais fácil de conseguir uma referência para um nó é através da variável que definimos quando criamos um nó. Assim como os nós, os relacionamentos também possuem um tipo.

O comando para criar um relacionamento é:

CREATE (no1)-[: TIPO_RELACIONAMENTO]->(no2).

Obs: “->” indica a direção do relacionamento.

Nas versões mais atuais do Neo4j, o *cypher* foi introduzido como *query language* oficial do banco de dados. A principal palavra chave (*keyword*) para uma *query* usando o *cypher* é o MATCH. Este comando é usado para definir a estrutura do grafo que estamos buscando. Além do MATCH, a outra parte obrigatória em uma *query* de busca é o RETURN, que é usado para definir o que a *query* retornará.

Exemplo de busca:

```
MATCH (x: TipoDoNo)
RETURN x.atributo
```

Desta forma, todos os nós do tipo "TipoDoNo" será armazenado na variável "x", depois somente o atributo que atende os requisitos do MATCH retornará.Uma das grandes vantagens de utilizar um banco orientado a grafo é que, além de retornar atributos, também podemos retornar nós inteiros, equivalente, a retornar um registro todo de uma tabela relacional. Com isso, aproveitamos a visualização do grafo.

```
MATCH       (x: TipoDoNo)
RETURN      x
```

Para editar o conteúdo dos nós partiremos do pressuposto que este já existe, então, a *query* ficará assim:

```
MATCH (nome_de_variavel: TipoDoNo {prop1: valor1})
SET nome_de_variavel.valor1 = 'valorX'
RETURN nome_de_variavel
```

Para remover um atributo podemos simplesmente atribuir o valor nulo a ele.

```
MATCH (nome_de_variavel: TipoDoNo {prop1: valor1})
SET nome_de_variavel.valor1 = 'null'
RETURN nome_de_variavel
```

E para deletar um nó adicionamos o DELETE.

```
MATCH (nome_de_variavel: TipoDoNo {prop1: valor1})
DELETE nome_de_variavel
```

Caso este nó possua uma relação, então é necessário apagar esta primeira para depois apagar o nó.

MATCH (nome_de_variavel: TipoDoNo {prop1: valor1}) – [rel] –()

DELETE rel

MATCH (nome_de_variavel: TipoDoNo {prop1: valor1})

DELETE nome_de_variavel

A função MATCH possui muitas outras utilidades que não abordaremos neste artigo. Mas dependendo do caso serão de suma importância.

- **MongoDB**

Para inserir um documento, se usa a função chamada *insert*. A função *insert* deve ser chamada a partir do nome de uma coleção, que deve ser chamada a partir do objeto base. Dessa forma: db.nome_da_colecao.insert(CORPO_DO_DOCUMENTO).

Onde o "Corpo do Documento" não precisa de um padrão ou formato, a fim de ilustrar melhor o comando anterior, um exemplo.

db.funcionarios.insert

{"nome": "Fulano",
"salário": "1000",
"dataAdmissao": new Date(2000, 1, 15),
"cargo": "Representante Técnico"}

- Obs: No campo "dataAdmissão" a data passada foi 15/02/2000, pois os meses começam em 0.

Com o documento inserido, para fazer buscas, o comando utilizado é o "*find*". O comando "*find*" funciona com critérios, como se fosse uma cláusula "WHERE" do SQL. Então uma busca no MongoDB fica assim:

db.nome_da_colecao.find{ "argumento": "nome"}

Desta forma, se quiséssemos buscar o documento inserido anteriormente o comando ficaria assim:

db.funcionarios.find{"nome": Fulano",

”dataAdmissao”:ISODate(“2000-01-15T00:00:00Z”),

”cargo”: “Representante Tecnico”}

Agora, os comandos básicos restantes no MongoDB são UPDATE e DELETE. Então atualizaremos nosso documento para depois excluí-lo.

db.funcionarios.update({“_id” : ObjectId (“SeuID”)},

{$set: {“nome”: “Ciclano”}})

- Onde SeuID é o ID gerado pelo MongoDB que você esta utilizando.
- Onde $set comando referente ao WHERE no SQL.

Para finalizar removeremos o objeto atualizado, com o seguinte comando:

db.funcionarios.remove({“nome” : “Ciclano”})

Com base nos comandos de cada tipo de banco e os resultados obtidos, podemos chegar as seguintes conclusões:

Tabela 1 –Comparativo entre os modelos NoSQL

	Vantagens	**Desvantagens**
Chave-Valor	Pesquisa Rápida	Dados armazenados não possuem *schema*
Colunas	Pesquisa Rápida, boa distribuição de armazenamento	API de baixo nível
Documentos	Tolerante a dados incompletos	*Query* não possui boa performance e sintaxe padrão
Grafos	Algoritmos Gráficos, conectividade, N grau de relacionamento	Percorre todo o gráfico para consultas, difícil agrupamento

Tabela 2 – Exemplos de usos indicados

	Quando Utilizar	**Quando não utilizar**
Chave-Valor	Armazenar informações de sessão, perfis de usuário, preferências, dados de carrinho de compras.	Relacionamentos entre dados, transações com múltiplas operações, consulta por dados, operações por conjuntos.
Colunas	Registros de eventos (log), Sistemas de gerenciamento de conteúdo, plataformas de blog, análise web ou tempo integral, e-	Consultas em estruturas agregadas variáveis, transações complexas com diferentes operações.

	commerce.	
Documentos	Registros de eventos (log), Sistemas de gerenciamento de conteúdo, plataformas de blog, contadores, uso por tempo determinado.	Quando as transações ACID são necessárias para gravação e leitura.
Grafos	Dados conectados, Roteamento, envio e serviços baseados em localizações, mecanismos de recomendação.	Aplicações que requerem atualização dos dados.

5. CONCLUSÃO

Conforme apresentado, apesar das diversas vantagens de cada tipo de banco NoSQL, existem várias desvantagens para os mesmos e cabe ao Administrador de Banco de Dados ou a equipe envolvida na escolha do mesmo, saber para cada situação, a melhor aplicação possível a ser utilizada. Sendo assim o NoSQL não substitui o SQL e sim o complementa de modo que existem casos onde ambos atuam em sincronia. O NoSQL ainda permite abordagens totalmente diferentes e muitas vezes mais práticas facilitando o uso e diminuindo a necessidade de recursos.

REFERÊNCIAS BIBLIOGRÁFICAS

LIMA, Claudio de, MELO, Ronaldo S. **Um Estudo sobre Modelagem Lógica para Bancos de Dados NoSQL**. Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (ERBD: Escola Regional de Banco de Dados. Florianópolis) - SC, 2015.

VIEIRA, Marcos Rodrigues, FIGUEIREDO, Josiel Maimone, LIBERATTI, Gustavo, VIEBRANTZ, Alvaro Fellipe Mendes.**Bancos de Dados NoSQL: Conceitos, Ferramentas, Linguagens e Estudos de Casos no Contexto de Big Data**. SBBD Simpósio Brasileiro de Banco de Dados. 2012.

TOTH, Renato Molina. **Abordagem NoSQL – uma real alternativa**. Universidade Federal de São Carlos. Sorocaba, SP.

HARRISON, Guy. **Next Generation Databases**. Apress,2015.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistemas de Bancos de Dados.** 4ªEd. Rio de Janeiro: Editora Elsevier, 2006.

PANIZ, David. **Como armazenar os dados de uma aplicação moderna**. Casa do Código, 2017.

FOWLER, Martin, SADALAGE, Pramod. **NoSQL Essencial – Um guia conciso para o mundo emergente da Persistência Poliglota**. 1ª Edição, Tradução: Acauan Fernandes, Novatec. 2013.