Capítulo 2

NoSQL e a Importância da Engenharia de Software e da Engenharia de Dados para o Big Data

Tassio Sirqueira e Humberto Dalpra

Abstract

The world has been producing a large amount of data currently. With the Internet of Things, we have a network of devices capable of collecting, transmitting and processing data and with this large amount of data, storing and processing these is the new challenge. Relational databases are being used by large corporations, however, the high data production implies relational databases, difficulties in scalability and performance, considering the respect of ACID properties and normal forms. We are currently working with data, often without a fixed structure, which culminates in the use of non-relational database management systems (DBMS), also known as NoSQL DBs. To better understand this paradigm change, it is necessary to characterize this data, extracting the main characteristics and defining when to use a relational DBMS or a NoSQL. This paper aims to explain the main characteristics of the NoSQL databases and to mix, theory and practice, to create a CRUD application using the Java programming language, which uses MongoDB, one of the main NoSQL DBMSs on the market. Also the new profiles of software engineers and data will be addressed, and what the Big Data is breaking, opposite paradigms already established.

Resumo

O mundo vem produzindo uma grande quantidade de dados atualmente. Com a internet das coisas, temos uma rede de dispositivos capazes de coletar, transmitir e processar dados e com essa grande quantidade de dados, o armazenamento e processamento destes é o novo desafio. Os bancos de dados relacionais vêm sendo utilizados por grandes corporações, entretanto, a produção elevada de dados imputam, aos bancos relacionais, dificuldades de escalabilidade e performance, considerando o respeito as propriedades ACID e as formas normais. Atualmente já trabalhamos com dados, muitas vezes sem

estrutura fixa, o que culmina no uso de sistemas gerenciadores de bancos de dados (SGBDs) não relacionais, também conhecidos como SGDBs NoSQL. Para melhor compreensão dessa mudança de paradigma, é necessário a caracterização destes dados, extraindo as principais características e definindo quando deve-se utilizar um SGBD relacional ou um NoSQL. Esse trabalho visa explicar as principais características dos bancos de dados NoSQL e miscigenando, teoria e prática, criar uma aplicação CRUD, utilizando a linguagem de programação Java, que utilize o MongoDB, um dos principais SGBDs NoSQL do mercado. Também serão abordados os novos perfis dos engenheiros de software e de dados, e o que o Big Data representa de rompimento, frente a paradigmas já consolidados.

2.1. Introdução

Quando iniciamos o desenvolvimento de um novo projeto de software, nos são imputadas tomadas de decisões importantes como sobre qual linguagem de programação será utilizada no projeto, qual o *framework* e reuso de código será abordado, além dos requisitos que são definidos pelos usuários. Um detalhe importante e pouco discutido em geral é com relação a persistência dos dados, onde é definindo qual o tipo de Sistema Gerenciador de Banco de Dados (SGBD) será utilizado, dentre as várias opções e disponibilidades existentes no mercado da informática. Apesar dos bancos relacionais serem os mais utilizados na maioria dos projetos, alguns detalhes devem ser considerados na tomada de decisão como, por exemplo, a flexibilidade dos dados e a escalabilidade da aplicação.

Com o crescimento da internet e a popularização do chamado "Internet das Coisas" (Gubbi *et al*., 2013), onde objetos físicos possuem tecnologias embarcadas e são capazes de comunicar entre si, passamos a possuir redes que coletam, transmitem e processam dados em grande quantidade. Desde portais de notícias, com milhares de acessos simultâneos, a sistemas de informação, em tempo real, ou aplicações de IoT (*Internet of things* – Internet das Coisas), o volume gerado e a dificuldade de manipulação desses dados crescem a cada dia. Além do grande volume de dados a serem armazenados, temos os desafios de extrair destes dados informações úteis. A forma de armazenamento implica diretamente em como manipular e na facilidade de cumprir com essa função.

Essa nova demanda de flexibilidade e escalabilidade das aplicações por conta do volume de dados, velocidade de processamento e/ou falta de estrutura fixa dos dados, em muitos casos emerge o uso de sistemas gerenciados de banco de dados não relacionais, também conhecidos como SGBDs NoSQL (Leavitt, 2010). NoSQL (*Not Only SQL)*, traduzido como 'Não apenas SQL', ressalta a importância da utilização deste para diferentes tipos de SGBD, promovendo a ferramenta ao armazenamento de dados, de acordo com a necessidade.

Atualmente existem diversos sistemas gerenciadores de banco de dados não relacionais, os quais são classificados de acordo com a forma em que os dados são

persistidos. Entre os principais tipos de SGBD NoSQL temos: i) Chave-valor; ii) Colunar; iii) Grafo; e iv) Documento, o qual será o principal foco na descrição deste trabalho.

Nos SGBDs NoSQL tipo Chave-valor, todos os registros compõem a mesma coleção (Tabela) e a única característica em comum entre os registros é uma chave única (*Hash*), sendo o modelo mais simples e fácil de implementar. São exemplos desse tipo de SGBD NoSQL o Riak[1], Redis[2], LevelDB[3] e RocksDB[4].

Nos SGBDs NoSQL tipo Coluna, todos os registros fazem parte da mesma tabela, sendo que as colunas variam de acordo com os registros. Esse tipo de banco é otimizado para colunas de leitura e gravação, ao contrário dos SGBDs relacionais que são linhas de dados. São exemplos desse tipo de SGBD o HBase[5], Cassandra[6] e SimpleDB[7].

Já nos sistemas de bancos NoSQL tipo Grafo, os registros são representados como nós e os relacionamentos como arestas, formando um grafo. São exemplos desse tipo de SGBD o Neo4j[8], GraphBase[9] e HyperGraphBase[10].

Nos SGBDs NoSQL tipo Documento, cada registro fica armazenado em um arquivo específico, geralmente em formato JSON (*JavaScript Object Notation*) ou XML (*eXtensible Markup Language*), dentro de uma coleção. Diferente dos SGBDs relacionais, o esquema dos documentos pode variar, permitindo maior flexibilidade e o tamanho do documento torna-se variável. Alguns exemplos desse tipo de SGBD são o Couchbase[11], CouchDB[12] e o MongoDB[13].

Cada tipo de SGBD NoSQL possui suas vantagens e desvantagens, de acordo com o seu tipo e sua implementação, uma lista completa de SGBDs NoSQL pode ser consultada em <http://nosql-database.org/>. Ao longo deste trabalho enfocaremos nos bancos NoSQL do tipo Documento, imputando o MongoDB. Atualmente o MongoDB é o SGBD NoSQL mais popular de todos, desenvolvido como um projeto de código aberto,

---

[1] Riak: Disponível em <http://basho.com/products/#riak>.
[2] Redis: Disponível em <https://redis.io/>.
[3] LevelDB: Disponível em <https://github.com/google/leveldb>.
[4] RocksDB: Disponível em <http://rocksdb.org/>.
[5] HBase: Disponível em <http://hbase.apache.org/>.
[6] Cassandra: Disponível em <https://cassandra.apache.org/>.
[7] SimpleDB: Disponível em <https://aws.amazon.com/pt/simpledb/>.
[8] Neo4j: Disponível em <https://neo4j.com/>.
[9] GraphBase: Disponível em <https://graphbase.ai/>.
[10] HyperGraphBase: Disponível em <http://www.kobrix.com/hgdb.jsp>.
[11] Couchbase: Disponível em <https://www.couchbase.com/>.
[12] CouchDB: Disponível em <http://couchdb.apache.org/>.
[13] MongoDB: Disponível em <https://www.mongodb.com/>.

multiplataforma e com suporte as principais linguagens de programação. É importante destacar que os SGBDs NoSQL não vieram para substituir os SGBDs relacionais, mas sim como alternativa para determinadas aplicações.

A Engenharia de Software (ES) atua na área de Software e Serviços (Lopes *et al*. 2005), onde os profissionais devem possuir uma visão que engloba a criação, teste, implantação e manutenção de sistemas de software. Também são visados novos sistemas computacionais para o mercado, sendo vital que estes profissionais tenham a habilidade de captar as necessidades dos clientes, sendo que estas são importantes para chegar-se a melhor solução para os problemas.

As soluções supramencionadas incluem aplicações para empresas tais como sistemas de informação, aplicações de software embarcado, dispositivos móveis ou dispositivos de IoT e sistemas para ambientes industriais e de automação.

O engenheiro de software atua fortemente na interação humano/computador. Em meio as equipes de trabalho, buscam um maior envolvimento com os negócios e ambiente dos clientes, afim de apresentar e implementar as aplicações e soluções necessárias. Sendo assim é importante a atualização de tais profissionais, tendo em vista que os mesmos tendem a buscar o que há de mais novo em relação as novas demandas que surgem no campo da computação. Essas atualizações incluem novas linguagens de programação, novos domínios, como por exemplo o de internet das coisas, e também na área de armazenamento de dados, visto o grande volume de dados manipulados pelas empresas atualmente, popularmente conhecidos como *Big Data*.

Assim, os profissionais devem conhecer as novas tecnologias, no intuito de sempre encontrar as melhores soluções para cada problema. As características dos novos engenheiros de software e de dados serão melhor discutidas à frente, dada que as mesmas são de extrema importância para auxiliar na compreensão exata do contexto. Na próxima seção abordaremos o papel da engenharia de dados e da engenharia de software na computação.

2.2. Engenharia de Dados e de Software

A Engenharia contempla a ciência e tecnologia para criação e construção de sistemas que sanem problemas. Pode-se definir a Engenharia de Dados como uma área da Engenharia dedicada a processar e tratar dados para aplicações que utilizarão *Big Data*, e o banco de dados vem a ser uma representação dinâmica do mundo real, onde os dados podem sofrer alteração temporal. Como descrito por Date (2004), um sistema de banco de dados é apenas um “sistema computadorizado de manutenção de registros”.

Já na Engenharia de Software, os projetos enfocam Teste e Validação de Sistemas, Qualidade de Software, Engenharia Reversa e Reengenharia de Software Orientadas a Objetos, Desenvolvimento de Software Orientado a Objetos, Engenharia de Requisitos

de Software, Paradigma Transformacional de Desenvolvimento, Ferramentas e Técnicas para desenvolvimento de software, Reutilização e Desenvolvimento de Software na Computação Ubíqua.

Sommerville (2011) explica que existem vários tipos de sistemas de software, que vão desde aplicações simples até aplicações complexas de alcance mundial, e todas as aplicações necessitam de engenharia, pois diferentes tipos de software exigem diferentes abordagens. Um sistema de software é desenvolvido a partir da integração entre a engenharia de dados e da engenharia de software.

Durante o desenvolvimento de um software projeta-se o modelo de dados, as classes e suas relações e, nesse instante, é importante que analistas e programadores tenham conhecimento sobre o futuro da aplicação. Sommerville (2011) destaca que o papel da engenharia de software é apoiar o desenvolvimento profissional do software, acimo do desenvolvimento pessoal.

A mudança de tecnologia em banco de dados pode ser um fator crítico em aplicações com alta carga de trabalho e para isso, os profissionais devem saber o impacto que suas escolhas têm sobre o ciclo de vida do software. A mudança do tipo de SGBD não só impacta o banco de dados como a modelagem do sistema como um todo, e daí a importância da comunicação entre a engenharia de dados e a engenharia de software.

Desenvolver uma aplicação que envolve o paradigma de banco não relacional pode ser um desafio para as empresas acostumadas a desenvolverem aplicações tradicionais e a comunicação entre as áreas é que determina o sucesso do desenvolvimento.

Na próxima seção abordaremos um comparativo entre os SGBDs Relacionais e NoSQL, de modo a facilitar e melhorar a compreensão dos termos, definições, vantagens e desvantagens de cada um.

2.3. Banco de Dados Relacional vs. Banco de Dados Não Relacional

Atualmente as empresas sofrem um crescente volume de dados, onde as aplicações devem ser preparadas para suportar esta grande demanda, focando em um tempo de resposta satisfatório, haja vista que há uma quantidade grande de usuários, o que culmina em mudanças na engenharia de dados e software.

Como apresentado por Elmasri (2005), um banco de dados armazena um conjunto de dados do mundo real, o qual é denominado minimundo. O mesmo é projetado, construído e populado com dados de uma aplicação específica.

Nos últimos anos os avanços da tecnologia e a demanda por soluções cada vez mais pujantes, tornaram obsoletas as soluções de banco de dados tradicionais, que em sua natureza permitiam o armazenamento e manipulação de dados apenas em formato

alfanumérico. Essa nova demanda exige o armazenamento de imagens, *streams* de áudio e vídeo, dados de informações geográficas entre outras formas.

Com o crescimento contínuo na geração de dados, esta elevada quantidade impacta diretamente na escolha de um SGBD a ser utilizado no desenvolvimento de um software. Neste momento os engenheiros de dados devem considerar os prós e contras de cada tipo de SGBD, tendo em vista que os banco de dados são essenciais para a sociedade moderna (Elmasri, 2005). Ainda segundo Elmasri (2005), todo banco de dados cria alguma abstração sobre os dados, sendo essa característica importante para definição de qual banco de dados será utilizado em uma solução de software ou sistema.

Atualmente os SGBDs mais utilizados são relacionais, tendo suas primeiras implementações comerciais datadas na década de 80. Entre os SGBDs relacionais mais populares do mercado, podemos destacar o Oracle[14], o SQL Server[15], o MySQL[16] e o PostgreSQL[17]. Esses SGBDs representam o banco de dados como uma coleção de tabelas relacionadas, compostas por atributos com tipos específicos, onde cada registro é uma tupla da tabela, visando a representação de dados do minimundo do problema.

Algumas características devem ser observadas no modelo de banco de dados relacional, as quais indicam as formas normais (normalização) do banco. Essa normalização do banco relacional, busca garantir que todos os dados armazenados não sejam multivalorados, não apresentem dependência funcional parcial ou transitiva. Isso faz com que o banco de dados cresça em número de tabelas e relacionamentos, o que no *Big Data* gera consequências.

Já na modelagem de bancos de dados orientados a documentos por exemplo, pode-se utilizar uma estrutura complexa para armazenar os dados, onde as informações podem ser agrupadas em um único documento, de modo a representar a visão dos dados para os negócios da empresa.

Diferente da modelagem de banco relacional, onde a mesma é realizada no início do projeto, e ainda muita das vezes não temos a visão completa do projeto, em um banco NoSQL há liberdade de esquema, de modo que as alterações degradem menos o software. Vale ressaltar que mesmo com essa flexibilidade, deve haver um padrão a ser seguido, levando em consideração as seguintes características:

- Coerência: refere-se a um documento ser compreensível se analisado individualmente, ou seja, não depender de outros documentos.
- Independência: refere-se a um documento possuir razão própria para existir.

[14] Oracle: Disponível em <https://www.oracle.com/>.
[15] SQL Server: Disponível em <https://www.microsoft.com/pt-br/sql-server/>.
[16] MySQL: Disponível em <https://www.mysql.com/>.
[17] PostgreSQL: Disponível em <https://www.postgresql.org/>.

- Isolamento: refere-se que a modificação de um dado documento, não deve implicar na modificação de outro.

Além disso, como abordado por Cattell (2011), o *Big Data* fez com que, nos últimos anos, as aplicações fossem projetadas para fornecer uma melhor escalabilidade horizontal, possibilitando que os dados dos sistemas fossem distribuídos entre vários servidores (nós). Contudo, os sistemas de banco de dados tradicionais (relacionais) possuem pouca capacidade de expansão horizontal. Essa característica é explicada pelo teorema CAP (Brewer 2000), onde em um banco de dados há a presença de uma partição de rede, vindo a ser necessário a escolha entre a consistência e a disponibilidade dos dados. Uma representação do teorema CAP visto é representada na Figura 2.1.

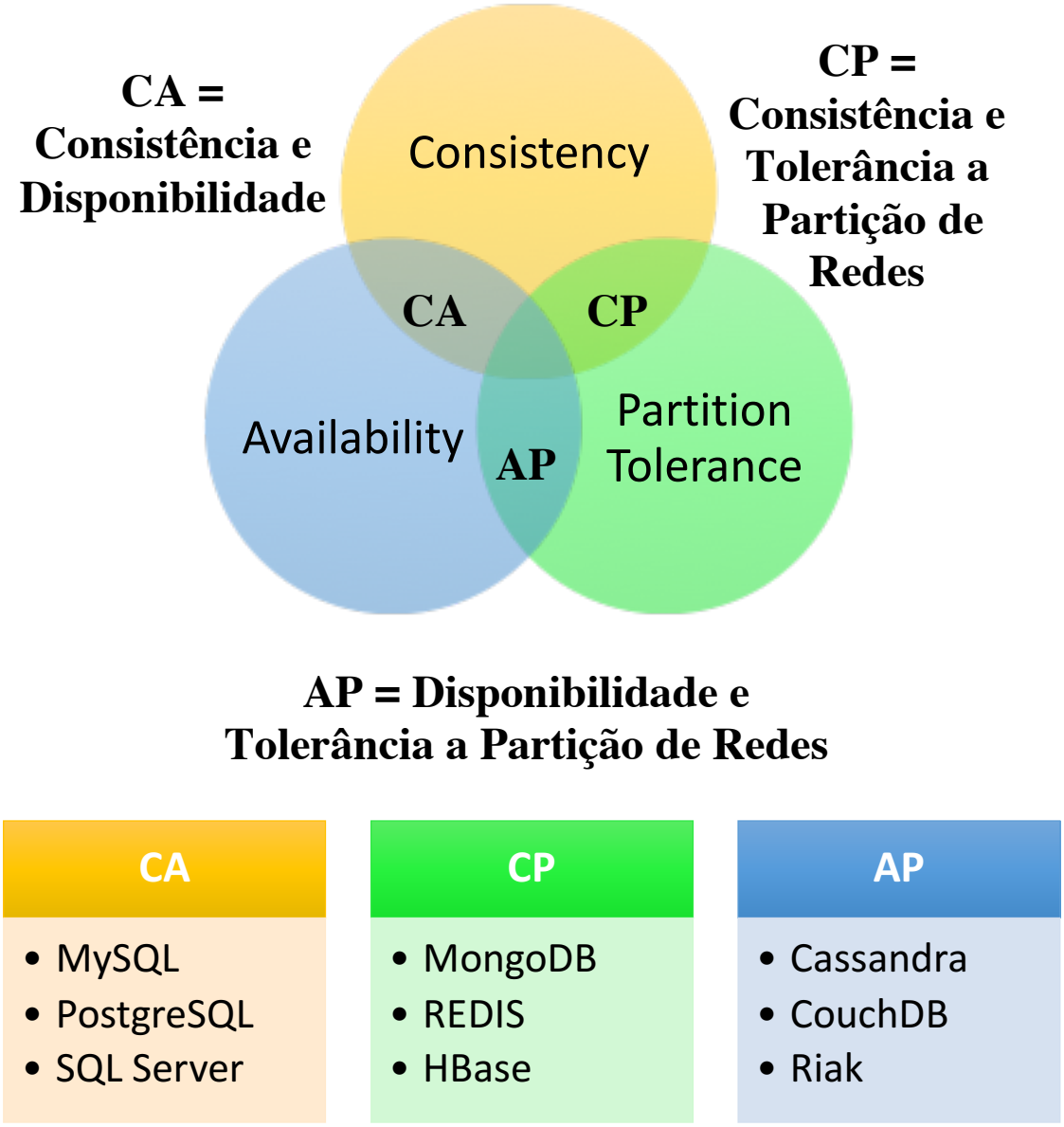

Figura 2.1. Representação do Teorema CAP.

O teorema CAP descreve o comportamento de um sistema distribuído, onde uma coleção de nós interconectados compartilha os dados entre si. Desta forma, um usuário pode conectar-se a qualquer nó desse sistema distribuído e realizar uma operação de escrita. Ao se conectar novamente ao mesmo ou outro nó do sistema distribuído, é possível realizar uma operação de leitura. A disponibilidade ou a consistência desse dado, dentro do sistema distribuído, é o que determina a reação do sistema distribuído.

Sendo assim, o teorema CAP afirma que um dado par de requisições, uma escrita seguida por uma leitura, em um panorama de sistemas distribuídos, garante apenas 2 entre os 3 comportamentos presentes no teorema. Esses comportamentos são definidos como:

- Consistência (Consistency): Consiste em garantir que a operação de leitura exiba o dado mais atualizado, independente do nó em que ele foi gravado. Essa característica garante que o cliente nunca receberá do banco um dado que já foi modificado.
- Disponibilidade (Availability): O comportamento de disponibilidade determina que nenhuma das requisições podem retornar erro e o usuário não pode aguardar indefinidamente pela resposta do SGBD, ou seja, o sistema deve se manter sempre disponível para leitura e gravação em um nó que não possui falha e este nó deve responder em um tempo razoável. Qualquer falha não pode afetar o funcionamento da aplicação que utiliza-a.
- Tolerância a Partição de Rede (Partition Tolerance): A Tolerância a Partição de Rede ou Tolerância a Falhas, determina que o sistema deve continuar operando mesmo após uma falha na rede (perda de conexão), garantindo que pelo menos as operações de leitura ocorram de forma normal.

Seguindo essa definição do teorema CAP, um SGBD pode combinar apenas duas dessas características, totalizando 3 combinações (CA, CP, AP). Essas combinações são explicadas na sequência:

- CA (Consistência e Disponibilidade): esta combinação têm sistemas de banco de dados que enfocam a consistência dos dados armazenados e a disponibilidade destes. Nesse modelo, qualquer falha em um dos nós, o sistema todo fica indisponível até que o nó que falhou volte ao normal. Essa é a configuração clássica dos SGBDs relacionais, sendo chamada de otimista, uma vez que a operação de escrita não gera inconsistência dos dados.
- CP (Consistência e Tolerância a Particionamento): Para sistemas que precisam da consistência forte e tolerância a particionamento, é necessário abrir mão da disponibilidade. Nesse modelo pode ocorrer que uma operação de escrita gere conflito entre os nós do particionamento de rede, sendo a disponibilidade comprometida até que ocorra um consenso entre os nós. Essa é uma abordagem

pessimista, visto que uma operação de escrita pode ser negada, apesar dos SGBDs tentarem evitar ao máximo que isso ocorra.

- AP (Disponibilidade e Tolerância a Particionamento): Quando pensamos em sistemas grandes e complexos, que atuam em todos os horários e dias (24/7) e nunca podem ficar *offline*, dependemos de alta disponibilidade e tolerância a particionamento. Esse modelo sacrifica a consistência, ou seja, o sistema sempre aceita as operações de escrita, mesmo que o sincronismo entre os nós ocorra em outro momento. Nesse período entre a persistência de um dado em um nó específico e seu sincronismo entre os demais nós do *cluster*, existe uma janela de inconsistência, onde uma operação de leitura em um nó que ainda não foi atualizado pode retornar dados desatualizados.

Enfatizando o ponto de vista do desenvolvimento, não existem tantas diferenças entre o CA e o CP, pois no modelo CA um sistema fica indisponível quando há particionamento, dado que possui alta disponibilidade por nó, enquanto no modelo CP o sistema tende a chegar em um consenso, onde o mesmo aceita uma escrita ou não. Na pior situação também é possível que esta fique indisponível para uma parte dos dados.

Essas características definem algumas vantagens ou desvantagens de um SGBD no sistema em que será empregado. Considerando um SGBD NoSQL, mesmo não existindo o particionamento, o SGBD pode priorizar o tempo de resposta e comprometer a consistência, dando prioridade a operações de leitura ao invés das operações de atualização, por exemplo.

O conhecimento do teorema CAP é fundamental para problemas do mundo real, onde a solução do problema atacado envolve o uso de sistemas distribuídos. Só assim os profissionais desse campo irão ter capacidade de desenvolver as melhores soluções.

A expansão do *Big Data* enfoca na necessidade de profissionais que saibam lidar com esse grande volume de dados, extraindo informações úteis e de valor para as empresas, sendo essa área denominada *Data Science* (ciência dos dados).

Data Science está cada vez mais em conjunto com *Big Data* e, conforme explicado por Dhar (2013), refere-se a buscar nos dados o conhecimento na forma de explicações testáveis e previsões sobre o universo analisado, ou seja, visa a extração de conhecimento para possíveis tomadas de decisões, podendo alinhar *Big Data*, aprendizado de máquina (*Machine Learning*) e mineração de dados (*Data Mining*).

Procurar nos dados padrões consistentes não é uma tarefa trivial, dado a volatilidade destes. Essa expansão do *Big Data* e o crescimento dos dispositivos de IoT, ocasionaram em operações com grandes quantidades de dados, muitas vezes semiestruturados, advindos de diferentes formas e formatos.

O objetivo da próxima seção é apresentar problemas reais que envolvem *Big Data* e como, parte do desafio para se encontrar a melhor solução para o problema, envolve o uso de SGBDs NoSQL.

2.4. *Big Data*, limitações dos SGBDs Relacionais e os novos desafios

Big data é uma realidade para muitas empresas, culminando em um desafio. O mercado demanda profissionais que estejam atualizados e saibam lidar com esse novo cenário. *Big Data* não é simplesmente trabalhar com grande volume de dados, mas saber trata-los e manipula-los em diversos formatos diferentes. Exemplos que podem ser citados são dados econômicos e de saúde, muitas vezes encontrados em arquivos de *Excel*, CSV (*Comma-separated values*) ou texto.

Como explicado por Mayer-Schonberger & Cukier (2014), o *Big Data* se baseia na capacidade de uma sociedade em obter informações de modo a gerar novas ideias, e agregar valor para bens e serviços, desafiando a maneira como vivemos e nossas decisões. Para Amaral (2016), todo dado possui um ciclo de vida e devemos extrair o máximo de informação enquanto o dado é valido. Esse ciclo de vida do dado é apresentado na Figura 2.2.

Essa capacidade de extrair informações do *Big Data,* envolve 3 características básicas, chamadas de 3Vs, que são: i) Volume; ii) Velocidade; e iii) Variedade. Também podem ser adicionadas outras características tais como, confiabilidade e valor. Esse processo de extração da informação, envolve a "análise descritiva", respondendo à pergunta: "O que aconteceu e por quê?"; a "análise preditiva", estimando a probabilidade de um dado evento e a "análise prescritiva" fornecendo recomendações (prescrições) específicas ao usuário.

Figura 2.2. Ciclo de vida do dado.

Essa diversidade de dados é algo crescente e até mesmo soluções simples podem se tornar interessantes para o uso de um SGBD NoSQL. Vamos descrever a seguir um exemplo simples de sistema e modelar o mesmo para um SGBD relacional e para um SGBD NoSQL, sendo realizado um comparativo da modelagem entre ambos.

Imagine um sistema onde será armazenado o cadastro de uma pessoa, contendo as seguintes informações básicas: i) nome; ii) sobrenome; iii) e-mail; iv) telefone e v) endereço. Agora imagine que esse sistema utilizará como SGBD um banco de dados relacional, o qual, no processo de modelagem, representará o resultado desse cadastro como uma tabela, conforme ilustrado na Figura 2.3. São exemplos de dados desta tabela do banco os valores exibidos na Tabela 2.1.

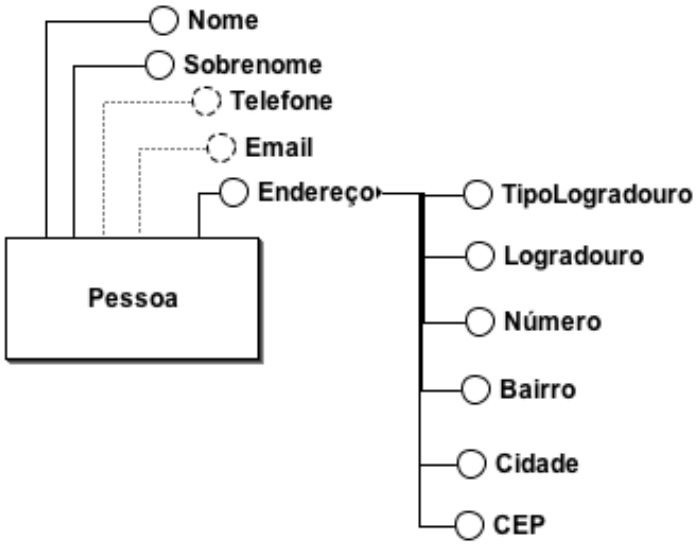

Figura 2.3. MER da Tabela Pessoa.

Tabela 2.1. Dados do Cadastro da Pessoa.

idPessoa	Nome	Sobrenome	E-mail	Telefone	Endereço
1	Tassio	Sirqueira	tassio @tassio.eti.br	3298444 0000	Rua A, 1, Centro, Matias Barbosa, 36120-000
2	Humberto	Dalpra	humbertodalpra @gmail.com	3298833 0000	Rua B, 1, Centro, Juiz de Fora, 36100-000
3	José	Da Silva		2199111 0000	Rua B, 1, Centro, Matias Barbosa, 36120-000
4	João	Nunes			Av Cardoso, 1, Borboleta, Juiz de Fora, 36110-100
5	Maria	Flor	flor.m @mail.com		Rua Bandeirantes, 1, Florais, Juiz de Fora, 36100-110

Conforme supramencionado, para os SGBDs relacionais, o banco de dados deve eliminar atributos multivalorados, a dependência funcional parcial e/ou transitiva,

também conhecida como formas normais ou processo de normalização. Isso representa que a Tabela 2.1 não pode permitir uma coluna chamada “endereço” contendo todos os dados, conforme representado.

Dessa forma, tem-se duas tabelas no sistema, uma para o cadastro da pessoa e outra para o seu endereço, conforme o modelo entidade-relacionamento apresentado na Figura 2.4.

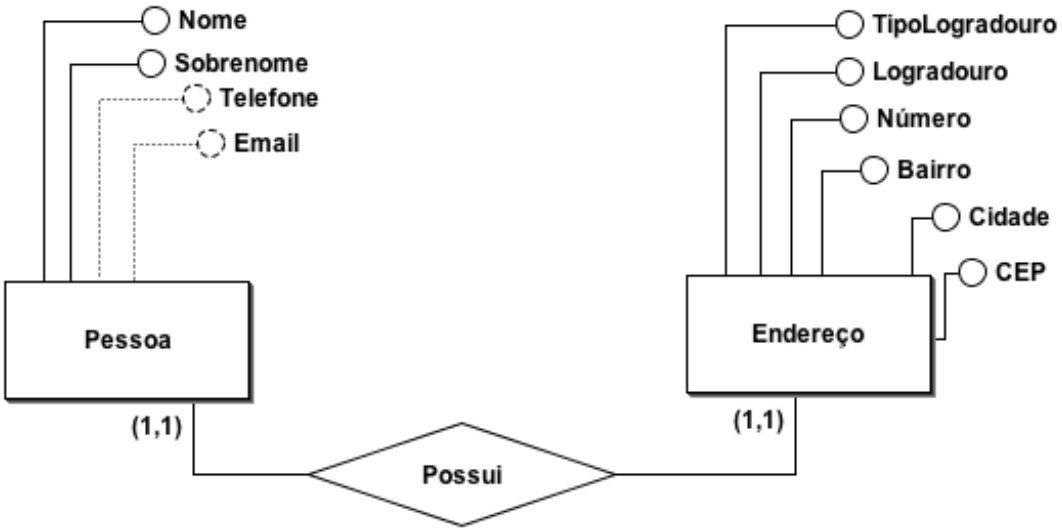

Figura 2.4. MER Pessoa e Endereço.

No exemplo exibido na Figura 2.4 resolvemos parte da normalização, tendo em vista que a tabela Pessoa receberá uma chave estrangeira da tabela Endereço. Dando continuidade. Imagine que esse mesmo cadastro possa permitir a pessoa possuir dois endereços, um residencial e outro comercial, e além disso, que possa ter dois e-mails (por exemplo, pessoal e de trabalho) e três telefones (por exemplo, de casa, celular pessoal e de trabalho).

Afim de sanar o problema do endereço passaríamos a chave estrangeira para a tabela Endereço e para os demais casos teríamos que criar novos atributos na tabela Pessoa. Se o usuário pudesse cadastrar quantos e-mails ou telefones ele deseja-se, isso não seria possível no modelo apresentado na Figura 2.4. Outro problema que pode-se visualizar na Tabela 2.1 é quando o cadastro não possui todos os atributos preenchidos, haja vista que nos SGBDs relacionais essas informações são representadas como vazias nas tuplas.

No modelo da Figura 2.5 tem-se uma representação de como os dados são armazenados em um SGBD NoSQL do tipo Documento, tendo em vista que cada documento pode apresentar uma estrutura de acordo com a necessidade. Assim, um exemplo dos dados completos, para o primeiro registro apresentado na Tabela 2.1, temos como saída o documento em formato JSON, apresentado na Figura 2.6.

O modelo de documento flexível do MongoDB facilita a criação de modelos de dados avançados que refletem como os dados são usados em um aplicativo. Entender como os dados são usados pelo seu aplicativo é fundamental para um *design* efetivo do banco de dados. Ao projetar um banco de dados de documentos, é importante entender quais entidades de aplicativo serão consultadas juntas. O MongoDB possibilita salvar dados comumente usados no mesmo registro. Ao salvar dados comumente usados em um único registro, os desenvolvedores de aplicativos podem obter enormes vantagens de desempenho, tendo apenas que consultar um único registro, em vez de criar consultas com várias associações.

Considerando essa flexibilidade, o exemplo da Tabela 2.1 para o 4º registro, o documento JSON gerado não necessita possuir os mesmos atributos do primeiro, conforme pode-se observar na Figura 2.7.

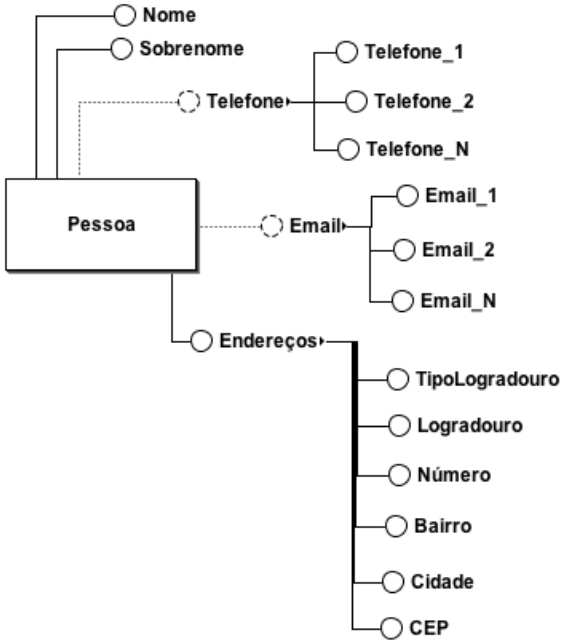

Figura 2.5. MER da tabela Pessoa estendido.

```
{
    "_id" : ObjectId("5aca8e88701e2721003eef97"),
    "nome" : "Tassio",
    "sobrenome" : "Sirqueira",
    "email" : "tassio@tassio.eti.br",
    "telefone" : "32984440000",
    "endereco" : {
        "tipoLogradouro" : "Rua",
        "logradouro" : "A",
        "numero" : "1",
        "bairro" : "Centro",
        "cidade" : "Matias Barbosa",
        "cep" : "36120-000"
    }
},
```

Figura 2.6. JSON armazenando os dados do primeiro registro.

```
{
    "_id" : ObjectId("5aca8e88701e2721003eef98"),
    "nome" : "João",
    "sobrenome" : "Nunes",
    "endereco" : {
        "tipoLogradouro" : "Av",
        "logradouro" : "Cardoso",
        "numero" : "1",
        "bairro" : "Borboleta",
        "cidade" : "Juiz de Fora",
        "cep" : "36110-100"
    }
}
```

Figura 2.7. JSON armazenando os dados do quarto registro.

Essa característica é importante pois, conforme supramencionado, se desejarmos adicionar outros atributos a um registro, podemos faze-lo ao documento ou a sub-atributos do documento sem alterar os demais registros, conforme demonstrado na Figura 2.8.

```
{
    "_id" : ObjectId("5aca912e701e2721288f1f6c"),
    "nome" : "Tassio",
    "sobrenome" : "Sirqueira",
    "email" : "tassio@tassio.eti.br",
    "telefone" : {
        "casa" : NumberLong("3232730000"),
        "celular" : 984440000
    },
    "endereco" : {
        "tipoLogradouro" : "Rua",
        "logradouro" : "A",
        "numero" : "1",
        "bairro" : "Centro",
        "cidade" : "Matias Barbosa",
        "cep" : "36120-000"
    }
}
```

Figura 2.8. JSON armazenando os dados do primeiro registro com dois telefones.

Essa dinamicidade no armazenamento dos dados que o banco NoSQL, do tipo documento, possui, permite armazenar dados com diferentes estruturas na mesma coleção. Apesar do exemplo demonstrado ser algo simples, quando trabalha-se com Big Data e dispositivos de IoT, sendo necessário armazenar esses dados, a variedade de dado e o tipo deles tendem a não manter-se constantes. Os dados em bases relacionais deixam de seguir a normalização ou o crescimento da base e o número de atributos vazios tornam-se enormes.

Analisando dados de saúde, por exemplo, advindos de um monitor multiparamétrico (utilizados por pacientes nas Unidades de Terapia Intensiva), observa-se que os dados recebidos podem variar conforme os sensores acoplados aos pacientes e aos alertas configurados.

Todos os equipamentos de saúde que atuam com o protocolo HL7, não seguem um padrão fixo de envio dos dados, o que dificulta seu armazenamento. Coletar dados de pacientes em UTIs que funcionam a todos os dias e horários (24/7) gera um *Big Data*, o qual não pode ser descartado, uma vez que os dados coletados devem ser mantidos pelos hospitais e servem como base de conhecimento para o corpo médico acompanhar o estado de saúde dos pacientes.

Áreas como saúde, economia, geologia e climatologia já enfrentam o problema do *Big Data*. Ao desenvolvermos aplicações para essas áreas, devemos identificar os pontos cruciais para o sucesso da solução. Na próxima seção abordaremos de modo mais profundo os SGBDs NoSQL, em especial o MongoDB.

2.5. Introdução ao NoSQL e MongoDB

Atualmente, banco de dados desenvolvidos com o uso do NoSQL permitem as empresas trabalharem com dados semiestruturados e com grande poder de escalabilidade para o universo do *Big Data*. Sistemas como o Facebook, que possuía em 2013 uma média de 2,4 bilhões de itens compartilhados por dia, entre os seus usuários, são dependentes de soluções NoSQL.

Essa quantidade e velocidade de produção dos dados, imputam em que, os mecanismos de armazenamento de dados, satisfaçam as necessidades de diferentes aplicações. Dentre as três soluções mais comuns pode-se citar o armazenamento direto no sistema de arquivos, bancos de dados relacionais e bancos de dados NoSQL. O termo "NoSQL" significa "*Not only SQL*" (não apenas SQL), onde neste pode-se aplicar o conceito *SQL* (*Structured Query Language* ou Linguagem de Consulta Estruturada).

Grolinger *et al*. (2013) discutem que o termo NoSQL foi usado pela primeira vez em 1998 para um banco de dados relacional, o qual omitiu o uso do SQL e foi retomado em 2009, quando passou a ser realmente utilizado por devido a necessidades que partiram de grandes instituições tais como Google, Yahoo, Facebook e Twitter. Grolinger *et al*. (2013) ainda afirmam que os sistemas de gerenciamento de banco de dados relacional, tradicionais, foram projetados em uma era em que o hardware disponível, bem como os requisitos de armazenamento e processamento, eram muito diferentes do que são hoje. Essa solução tem encontrado muitos desafios para atender aos requisitos de desempenho e dimensionamento do chamado "*Big Data*".

Algumas das razões pelas quais os SGBDs NoSQL vêm ganhando mercado, são explicadas por Strauch *et al.,* (2011), os quais abordam que os SGBDS NoSQL apresentam características importantes para o *Big Data*, sendo elas: i) evitar a complexidade desnecessária; ii) alta taxa de transferência; iii) escalabilidade horizontal; iv) evitar o mapeamento objeto-relacional que pode custar caro; v) a complexidade e o custo da configuração de *clusters* de banco de dados; vi) melhor desempenho, entre outras características.

Devido à variedade das abordagens NoSQL e a sobreposições em relação aos requisitos não-funcionais e ao conjunto de recursos, pode ser difícil obter e manter uma visão geral dos bancos de dados não relacionais. Diferente dos SGBDs relacionais, no NoSQL não há suporte ao ACID (*Atomicity, Consistency, Isolation, Durability)*. Também não há dependência de tabelas e colunas fixas, bem como não há o uso padrão das consultas SQL.

Dentre os projetos NoSQL mais notáveis até o momento, podemos citar o projeto de software livre MongoDB, que nada mais é que um banco de dados orientado a documentos, o qual armazena dados em coleções de documentos semelhantes a JSON, compostos por nomes de campos e um tipo específico de valor.

A distinção do SGBD MongoDB em relação a outros bancos de dados NoSQL é a poderosa linguagem de consulta baseada em documento, a qual torna fácil a transição de um banco de dados relacional para o MongoDB, haja vista que as consultas ou instruções SQL são convertidas à respectiva função. Ele também apresenta simplicidade na instalação e utilização.

Outro detalhe importante, conforme supramencionado, é que banco de dados relacionais permitem a escalabilidade, contudo, quanto maior o tamanho do banco de dados, maior é o custo de manter esta, seja para adicionar novas máquinas ou para manter a equipe responsável pela administração do banco. Já os bancos de dados não relacionais permitem uma escalabilidade mais barata e menos trabalhosa, visto que as máquinas não precisam ser poderosas e a equipe de administração do banco necessita de um número menor de pessoas. Tais características são associadas ao teorema CAP.

Na próxima seção será apresentado o banco de dados NoSQL MongoDB e suas principais características.

2.5.1. Apresentando o MongoDB

Banco de Dados Orientados a Documentos contém todas as informações importantes de uma entidade em um único documento, cada grupo de entidades formam as coleções características, tais como ser livre de esquemas, possuir identificadores únicos universais (*UUID*), possibilitar a consulta de documentos através de métodos avançados de agrupamento e filtragem (*MapReduce*), até permitir redundância e inconsistência, são os que definem essa classe de SGBD.

Esses bancos de dados orientados a documentos também são chamados de Bancos NoSQL (*Not Only SQL*). O termo NoSQL é devido à ausência do SQL (*Structured Query Language*), mas esse tipo de Banco de Dados não se resume apenas a isso. Alguns chegaram a defender o termo NoREL (*Not Relational*), o qual não foi muito recebido. Sintetizando o conceito, esse tipo de Banco de Dados não embute as ideias do modelo relacional e nem a linguagem SQL. Podemos citar que uma diferença fundamental entre

os dois modelos surge na criação de relacionamentos nos bancos de dados relacionais, os quais diferem dos bancos orientados a documentos, os quais não fornecem relacionamentos entre documentos. Os bancos de dados de documentos integram esses dados ao próprio documento.

Atualmente existem diversos Banco de Dados NoSQL, tais como os que foram apresentados na seção 2.1. O MongoDB é um SGBD voltado à alta performance, sem esquema fixo e orientado a documentos. Os documentos englobam formato BSON (*Binary JSON*), semelhante ao JSON, possibilitando que a estrutura e os campos do documento variem de uns para os outros ao longo do tempo.

O BSON é uma extensão do JSON, codificando de forma binária os documentos. Três características do formato BSON são: i) leveza; ii) transportabilidade; e iii) eficiência. Cada documento BSON permite a cada atributo possuir um tipo especifico de valor, seguindo a Tabela 2.2.

Tabela 2.2. Formatos suportados pelo BSON.

Tipo	Número	Apelido	Nota
Real	1	“double”	
Texto	2	“string”	
Objeto	3	“object”	
Vetor	4	“array”	
Dado binário	5	“binData”	
Indefinido	6	“undefined”	Depreciado
ID Objeto	7	“objectId”	
Booleano	8	“bool”	
Data	9	“date”	
Nulo	10	“null”	
Expressão regular	11	“regex”	
DBPointer	12	“dbPointer”	Depreciado
JavaScript	13	“javascript”	
Símbolo	14	“symbol”	Depreciado
JavaScript com escopo	15	“javascriptWithScope”	
Inteiro de 32-bit	16	“int”	

TimeStamp	17	“timestamp”	
Inteiro de 64-bit	18	“long”	
Decimal128	19	“decimal”	A partir da 3.4
Chave mínima	-1	“minKey”	
Chave máxima	127	“maxKey”	

O MongoDB é um banco de dados distribuído, com escalabilidade horizontal focada na distribuição geográfica, fornecendo mecanismos para consultas *ad hoc*, os quais contemplam indexação e agregação em tempo real. Ele possui uma API (*Application Programming Interface*) própria para o gerenciamento das transações no banco, onde geralmente os dados são manipulados em memória RAM, buscando maior velocidade nas operações de uma aplicação CRUD (*Create, Read, Update* e *Delete*). Tal API está disponível para o Java em seu repositório no GITHUB[18].

O MongoDB versão *Community* está disponível para instalação nos principais sistemas operacionais, sendo eles o Debian 7+, Ubuntu 12.04+, Red Hat/CentOS 6+, OS X 10.7+ e o Windows 7/Server 2008 R2+. O mesmo também possui drivers para as principais linguagens de programação do mercado, tais como o C, C++, Java, C#, Python, PHP, entre outras.

Atualmente, conforme pesquisa do StackOverflow 2017[19], o banco de dados MongoDB foi o mais votado dos bancos NoSQL em utilização, ocupando a 5ª posição do ranking geral de banco de dados. Nesta mesma pesquisa o banco ocupou a 3ª e 1ª posição no ranking, o qual considera as categorias de banco de dados mais amados e mais desejados respectivamente. Isso demostra o interesse da comunidade pelo banco MongoDB e como o mesmo vem crescendo em utilização e público.

A versão atual, estável, é a 3.6. Para a versão 4.0 Horowitz (2018) anunciou que o banco de dados adicionará suporte a transações com vários documentos, tornando-o o único banco de dados a combinar velocidade, flexibilidade e poder do modelo de documento com garantias ACID. Essa modificação trará ao banco mais familiaridade com os desenvolvedores que já estão acostumados a trabalhar com os tradicionais bancos de dados relacionais.

Na próxima seção abordaremos a instalação e preparação do servidor para uso.

[18] MongoDB Java Driver: Disponível em <https://mongodb.github.io/mongo-java-driver/>.

[19] Pesquisa StackOverflow 2017: Disponível em <https://insights.stackoverflow.com/survey/2017>.

2.5.2. Preparando o Ambiente

Ao abordarmos a utilização do MongoDB como banco de dados, é importante enfatizar que algumas configurações devem ser feitas durante e após a sua instalação.

A instalação do MongoDB pode ser realizada mediante ao *download* do mesmo em <http://www.mongodb.org/downloads>, devendo-se optar pela versão do sistema operacional em que será utilizado o SGBD, ou ainda via 'apt', 'yum' ou 'brew' no caso das distribuições GNU/Linux ou Mac OS X.

Para o sistema operacional Windows é disponibilizado um instalador MSI. Já para o Mac OS X e GNU/Linux, temos como opção o SGBD compactado em TGZ, além das opções supracitadas (brew, apt ou yum). Mais informações sobre a forma de instalação para cada sistema operacional está disponível no endereço <https://docs.mongodb.com/master/administration/install-community/>. Nessas instalações são ofertados 5 pacotes, conforme demonstrado na Tabela 2.3.

Tabela 2.3. Pacotes que compõem o MongoDB.

Pacote	Descrição
mongodb-org	Um meta-pacote que instalará automaticamente os pacotes de quatro componentes listados abaixo.
mongodb-org-server	Contém o daemon mongod e os scripts de configuração e de inicialização.
mongodb-org-mongos	Contém o daemon mongos*.
mongodb-org-shell	Contém a interface de acesso ao mongo.
mongodb-org-tools	Contém as seguintes ferramentas do MongoDB: mongoimport bsondump, mongodump, mongoexport, mongofiles, mongorestore, mongostat e mongotop.

O *'mongos' é um serviço de roteamento para configurações de compartilhamento do MongoDB, que processa consultas da camada de aplicação e determina o local desses dados no *cluster* fragmentado. Da perspectiva do aplicativo, uma instância do 'mongos' se comporta de maneira idêntica a qualquer outra instância do MongoDB.

O SGBD é configurado via passagem de parâmetros na inicialização ou via arquivo de configuração (mongod.conf ou mongod.cfg, para GNU/Linux e OS X ou Windows). Nesse arquivo de configuração são especificados os valores para o destino dos logs do SGDB e onde os dados serão armazenados (dbpath), entre outras configurações possíveis. O mesmo ocorre para o serviço 'mongos'. O requisito básico que deve ser especificado na iniciação do 'mongod' (serviço do MongoDB) é o caminho

do banco. Uma especificação completa do arquivo de configuração do SGBD, pode ser encontrada em sua documentação[20].

Por padrão o SGBD utiliza a porta '27017' para comunicação, via *shell* ou por meio de alguma ferramenta IDE, como o 'Studio 3T'[21] ou o 'NoSQLBooster'[22], o qual será utilizado ao longo deste trabalho. Contudo, após a instalação devemos realizar as configurações de segurança para o administrador e criar os novos usuários e seus respectivos bancos.

O SGBD MongoDB, por padrão, não possui nenhum usuário ou senha para acesso aos bancos, o qual deve ser configurado logo após a instalação. Para isto, podemos acessar o SGBD utilizando a IDE 'NoSQLBooster', especificando a URI de conexão "mongodb://localhost:27017". Onde 'localhost' pode ser substituído pelo endereço IP do servidor, caso não seja a máquina local.

Para criarmos usuários no MongoDB, devemos utilizar a função 'db.createUser()'. Para a criação do usuário devemos informar o nome, a senha e os *roles* (papéis). Em *roles* devemos informar quais os privilégios para o usuário e em qual base de dados ele terá tais privilégios. Os *roles* possíveis para os privilégios em um usuário administrador de um *cluster* são apresentados a Tabela 2.4:

Tabela 2.4. *Roles* para administrador do *cluster*.

Papel	Descrição
clusterAdmin	É o maior privilégio para usuários administradores do *clusters*. Inclui todos os privilégios que serão listados abaixo e também possui o privilégio da ação de 'dropDatabase'.
clusterManager	Fornece o gerenciamento e monitoramento das ações no *cluster*. Um usuário com esse papel pode acessar as configurações de bases locais, que são usadas em réplicas e fragmentação.
clusterMonitor	Fornece apenas leitura para ferramentas de monitoramento no *cluster*.
hostManager	Permite monitorar e gerenciar os servidores.

Outras regras importantes são referentes ao gerenciamento das bases de dados, onde podemos destacar as regras apresentadas na Tabela 2.5.

[20] Documentação do MongoDB. Disponível em <https://docs.mongodb.com/manual/reference/configuration-options/>.
[21] Studio 3T. Disponível em <https://studio3t.com/>.
[22] NoSQLBooster. Disponível em <https://nosqlbooster.com/home>.

Tabela 2.5. *Roles* para administrador da base de dados.

Papel	Descrição
root	Fornece acesso às operações e a todos os recursos das funções 'readWriteAnyDatabase', 'dbAdminAnyDatabase', 'userAdminAnyDatabase', funções 'clusterAdmin', restauração e backup do sistema.
userAdminAnyDatabase	Define as mesmas permissões que o 'userAdmin', contudo aplicadas a qualquer base de dados.
dbAdminAnyDatabase	Fornece o mesmo acesso às operações de administração do banco de dados que o 'dbAdmin' para todas as bases de dados.
userAdmin	Permite a esses usuários criar e modificar configurações e determinar usuários para um banco de dados.
dbAdmin	Define que o usuário será o administrador de uma base de dados específica.
dbOwner	Define que o usuário será o proprietário da base de dados e terá permissão administrativa da mesma. Essa regra combina os privilégios concedidos pelas funções 'readWrite', 'dbAdmin' e 'userAdmin'.
readWrite	Permite ao usuário as operações de leitura e gravação na base de dados.
read	O usuário terá a permissão de somente leitura na base de dados.

Essas regras são definidas na base de dados "admin" do SGBD, sendo que uma lista completa contendo outras regras está disponível no site da documentação[23]. Para definirmos um usuário super-administrador para o SGBD, podemos especificá-lo com suas regras, conforme demonstrado na Figura 2.9.

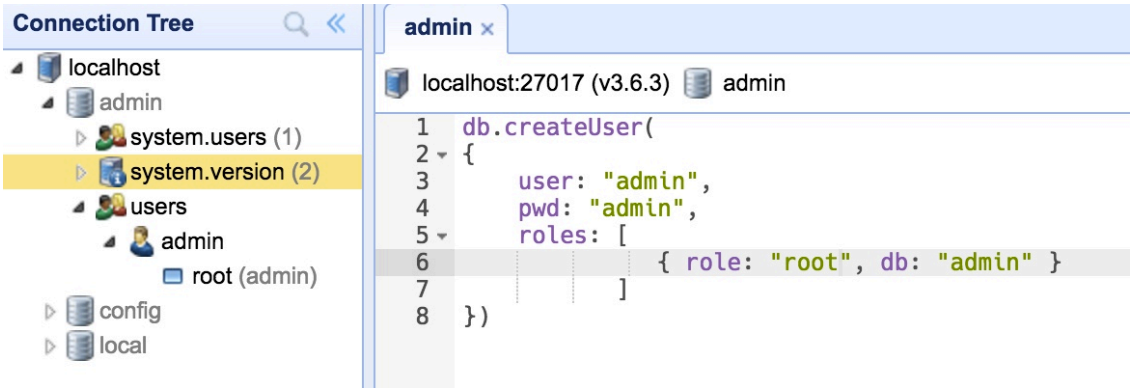

Figura 2.9. Usuário super-administrador do SGBD.

[23] Permissões do SGBD: Disponível em <https://docs.mongodb.com/manual/reference/built-in-roles/>.

Após configurarmos o super-administrador devemos criar um banco de dados e um usuário responsável por este banco, conforme demostra a Figura 2.10.

Figura 2.10. Usuário proprietário da base JAI.

Para que as novas regras de autenticação passem a ser aplicadas pelo SGBD, é necessário reiniciar o serviço do banco passando como parâmetro, nas configurações, a opção '--auth'. Assim o SGBD exigirá que qualquer operação seja autenticada por algum usuário que possua à devida permissão.

Além da opção para adição de usuários ao SGBD, temos as opções para remoção ('db.dropUser()' e 'db.dropAllUsers()'), gerência de permissões ('db.grantRolesToUser()', 'db.revokeRolesFromUser()' e 'db.auth()') e de atualizações dos usuários ('db.updateUser()', 'db.changeUserPassword()', 'db.getUser()' e 'db.getUsers()'). A utilização destas funções pode ser encontrada em <https://docs.mongodb.com/manual/reference/method/js-user-management/>.

Na próxima seção será iniciado o desenvolvimento de uma aplicação Java, onde serão demostrados as formas de conexão com o SGBD e os primeiros passos para manipulação do banco de dados.

2.5.3. Criando o Primeiro Exemplo

Mediante a configuração do SGBD MongoDB, criação de um usuário e banco de dados para utilização, chegamos no momento de criar o primeiro exemplo. Para este utilizaremos uma aplicação Java, pela qual será demonstrado como realizar a conexão com o SGBD, a modelagem do sistema e como fazemos a manipulação dos dados no Java, o qual está direcionado para o MongoDB.

A aplicação consistirá em um CRUD, demostrando a criação, leitura, atualização e deleção de dados em um banco de dados NoSQL (MongoDB). Essa aplicação consiste em armazenar dados de um autor, o qual terá os mesmos dados da Tabela 2.1, e de suas publicações. O sistema será composto de duas coleções, uma contendo os dados dos autores e outra contendo as publicações.

As conversões dos dados de entrada serão inicialmente modeladas de forma similar ao HashMap, utilizando o 'BasicDBObject', o qual converte os dados para BSON, antes de sua gravação no banco e posterior recuperação.

Para iniciar o desenvolvimento da aplicação são necessárias as APIs do MongoDB, conforme já mencionado, em especial a "bson", "mongodb-driver" e "mongodb-driver-core", atualmente na versão 3.6.3. Essas bibliotecas podem ser importadas para o projeto via Maven[24] ou manualmente.

Importadas as bibliotecas ao projeto podemos definir a conexão com a base de dados, conforme Figura 2.11.

```
MongoClientURI mongoURI = new MongoClientURI("mongodb://jai:jai@localhost:27017/?authSource=JAI");
MongoClient mongodb = new MongoClient(mongoURI);
MongoDatabase db = mongodb.getDatabase("JAI");
```

Figura 2.11. Conexão com o MongoDB.

Conforme a Figura 2.11, na primeira linha temos o URI (*Uniform Resource Identifier*) onde são passados o nome do usuário do banco (jai), a senha deste usuário (jai), o host onde está sendo executada a instancia do MongoDB (localhost), a porta de conexão com o SGBD (27017) e a base em que o usuário irá autenticar-se (JAI). A segunda linha da Figura 2.11 é responsável por realizar a conexão. Na última linha é definida a base de dados que será utilizada pela aplicação (JAI).

Um 'BasicDBObject' é um objeto no formato BSON, que pode conter uma única informação por campo, uma lista ou um outro objeto BSON. Um exemplo da construção desse tipo de objeto pode ser visto na Figura 2.12.

```
BasicDBObject pessoa = new BasicDBObject();
pessoa.put("nome", "Humberto");
pessoa.put("sobrenome", "Dalpra");
pessoa.put("email", "humbertodalpra@gmail.com");

BasicDBObject tel = new BasicDBObject();
tel.put("celular", Long.parseLong("32988330000"));
pessoa.put("telefone", tel);

BasicDBObject end = new BasicDBObject();
end.put("tipoLogradouro", "Rua");
end.put("logradouro", "B");
end.put("numero", "1");
end.put("bairro", "Centro");
end.put("cidade", "Juiz de Fora");
end.put("cep", "36100-000");

pessoa.put("endereco", end);
```

Figura 2.12. Objeto BSON.

Neste exemplo da Figura 2.12 tem-se um objeto BSON 'pessoa', o qual contém o nome, sobrenome, e-mail, telefone e endereço. Entretanto, o telefone e o endereço também são objetos BSON que compõem o objeto pessoa.

[24] Driver MongoDB para Java: Disponível em <https://mongodb.github.io/mongo-java-driver/>.

Cada objeto BSON é armazenado na sua devida coleção, onde são agrupados documentos similares. A especificação da coleção e o salvamento do objeto BSON (pessoa) criado são apresentados na Figura 2.13.

```
MongoCollection<BasicDBObject> colecaoPessoa = db.getCollection("Pessoa", BasicDBObject.class);
colecaoPessoa.insertOne(pessoa);
```

Figura 2.13. Especificação da coleção e salvamento do Objeto BSON.

Além do método 'insertOne', responsável por inserir um documento em uma coleção, temos métodos para remoção, deleção, busca e atualização, os quais podem operar sobre um único documento ou um conjunto de documentos, conforme será apresentado na próxima seção, onde também será explicada a manipulação de documentos do SGBD MongoDB.

2.5.4. Manipulando as Informações

Algumas definições são importantes para a manipulação de dados, como por exemplo os conceitos vinculados ao SGBD relacional e como são representados no SGBD NoSQL. Fazendo uma associação entre os SGBDs relacionais e os SGBDs NoSQL do tipo documento, temos uma representação, a qual é apresentada na Figura 2.14.

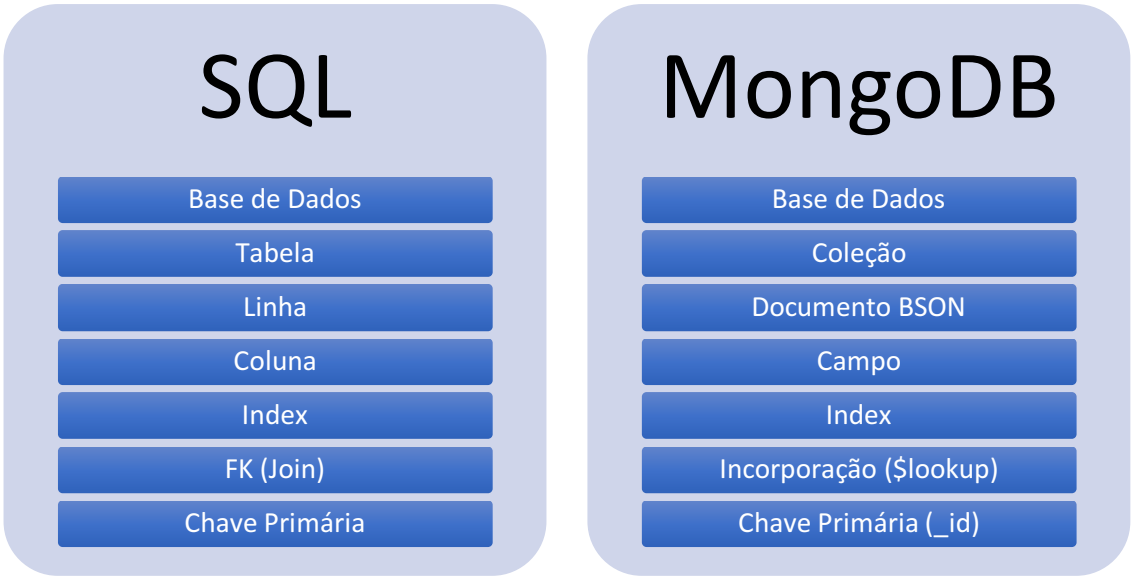

Figura 2.14. Conceitos SQL e NoSQL.

Nessa associação temos que o conceito de base de dados é igual a ambos os tipos de SGBD. Já o conceito de tabela dos SGBDs relacionais é substituído pelo de coleção, visto que ao invés de linhas dos dados temos agora documentos em formato BSON. Cada coluna passa a ser representada como um campo do documento BSON, que pode conter um único valor, uma lista de valores ou um objeto, conforme explicado na seção anterior.

Nos bancos de dados NoSQL, do tipo documento, não existe o conceito de chave estrangeira, contudo é possível realizar a incorporação de documentos entre diferentes

coleções, onde essa definição será explicada na seção 2.6.1 e, por fim, a chave primaria equivale ao campo '_id' do documento BSON.

Prosseguindo com o desenvolvimento da aplicação CRUD, a qual vem usando a linguagem de programação Java, abordaremos na sequência a manipulação dos dados em banco, considerando a persistência de um lote de documentos, a busca de um registro único e a busca de múltiplos registros, atualização de um registro possibilitando alteração de estrutura, entre outros métodos importantes. Uma lista dos principais métodos de manipulação de dados do SGBD MongoDB pode ser observado na Tabela 2.6.

Tabela 2.6. Métodos de manipulação de documentos.

Nome	Descrição
insertOne()	Insere um novo documento em uma coleção.
insertMany()	Insere um conjunto de novos documentos em uma coleção.
insert()	Cria um novo documento em uma coleção.
updateOne()	Modifica um único documento em uma coleção.
updateMany()	Modifica um conjunto de documentos em uma coleção.
update()	Modifica um documento em uma coleção.
deleteOne()	Exclui um único documento em uma coleção.
deleteMany()	Exclui um conjunto de documentos em uma coleção.
remove()	Exclui documentos de uma coleção.
find()	Executa uma consulta em uma coleção ou visão e retorna um cursor de objetos.
findOne()	Executa uma consulta e retorna um único documento.
findOneAndDelete()	Encontra um único documento e o apaga.
findOneAndReplace()	Encontra um único documento e o substitui.
findOneAndUpdate()	Encontra um único documento e o atualiza.

Demonstrando o uso de algumas dessas funções no CRUD Java, além do método 'inserOne', apresentado na seção anterior, o método 'find' permite listar uma coleção e retornar todos os seus documentos, caso não seja incorporada nenhuma condição. Seu uso pode ser visto na Figura 2.15.

```
MongoClientURI mongoURI = new MongoClientURI("mongodb://jai:jai@localhost:27017/?authSource=JAI");
MongoClient mongodb = new MongoClient(mongoURI);
MongoDatabase db = mongodb.getDatabase("JAI");
MongoCollection<BasicDBObject> colecaoPessoa = db.getCollection("Pessoa", BasicDBObject.class);

for (BasicDBObject doc : colecaoPessoa.find()) {
    System.out.println(doc);
}
```

Figura 2.15. Listagem de uma coleção no MongoDB.

Já o método 'updateOne' permite atualizar um único documento, onde esta atualização pode incluir ou não mudança em sua estrutura. Nesse método são passados dois parâmetros, um que atuará como uma *query* de busca e outro que serão os novos valores do Documento BSON. Um exemplo de atualização utilizando Java é apresentado na Figura 2.16.

```
MongoClientURI mongoURI = new MongoClientURI("mongodb://jai:jai@localhost:27017/?authSource=JAI");
MongoClient mongodb = new MongoClient(mongoURI);
MongoDatabase db = mongodb.getDatabase("JAI");
MongoCollection<BasicDBObject> colecaoPessoa = db.getCollection("Pessoa", BasicDBObject.class);

BasicDBObject query = new BasicDBObject();
query.put("nome", "João");

BasicDBObject pessoa = new BasicDBObject();
pessoa.put("nome", "João");
pessoa.put("sobrenome", "Silva");

BasicDBObject end = new BasicDBObject();
end.put("tipoLogradouro", "Av");
end.put("logradouro", "Cardoso");
end.put("numero", "1");
end.put("bairro", "Borboleta");
end.put("cidade", "Juiz de Fora");
end.put("cep", "36110-100");
pessoa.put("endereco", end);

BasicDBObject update = new BasicDBObject();
update.put("$set", pessoa);

colecaoPessoa.updateOne(query, update);
```

Figura 2.16. Atualização de um documento BSON em Java.

Nesse exemplo de atualização, observe que passou-se a *query* e no update indicou-se o '$set', que é um operador que substitui o valor de um campo pelo novo valor especificado. Por fim, uma aplicação CRUD deve conter um método de deleção de documentos, o qual pode ser visto na Figura 2.17. Para a remoção de um documento é especificado a condição que o documento deve possuir.

```
MongoClientURI mongoURI = new MongoClientURI("mongodb://jai:jai@localhost:27017/?authSource=JAI");
MongoClient mongodb = new MongoClient(mongoURI);
MongoDatabase db = mongodb.getDatabase("JAI");
MongoCollection<BasicDBObject> colecaoPessoa = db.getCollection("Pessoa", BasicDBObject.class);

BasicDBObject query = new BasicDBObject();
query.append("nome", "João");
colecaoPessoa.deleteOne(query);
```

Figura 2.17. Remoção de um documento BSON em Java.

Além das funções de manipulação de dados apresentadas no CRUD, temos as de manipulações de coleções e as de busca condicionadas, onde são especificados filtros para retorno dos resultados. Para apresentarmos essa manipulação de coleções e as buscas condicionadas, considera-se os dados da Tabela 2.7, representando uma tabela de um SGBD relacional e a Figura 2.18, representando os mesmos dados em BSON.

Tabela 2.7. Tabela 'Publicacoes' em um SGBD relacional.

IdPublicacoes	Autor	Artigo
1	Humberto & Tassio	Using Ontology and Data Provenance to Improve Software Processes
2	Tassio	A Software Framework for Data Provenance
3	Stonebraker	Newsql: An alternative to nosql and old sql for new oltp apps

```
/* 1 createdAt:17/04/2018 09:59:27*/
{
    "_id" : ObjectId("5ad5efaf90e8f81d50f67d00"),
    "artigo" : "Using Ontology and Data Provenance to Improve Software Processes",
    "autor" : [
        "Tassio",
        "Humberto"
    ]
},

/* 2 createdAt:17/04/2018 09:59:27*/
{
    "_id" : ObjectId("5ad5efaf90e8f81d50f67d01"),
    "artigo" : "A Software Framework for Data Provenance",
    "autor" : "Tassio"
},

/* 3 createdAt:17/04/2018 09:59:28*/
{
    "_id" : ObjectId("5ad5efb090e8f81d50f67d02"),
    "artigo" : "Newsql: An alternative to nosql and old sql for new oltp apps",
    "autor" : "Stonebraker"
}
```

Figura 2.18. Dados da coleção 'Publicacoes' em BSON.

A Tabela 2.8 apresenta uma associação entre as *querys* SQL e a mesma para o SGBD MongoDB, evidenciando as diferenças, onde a coluna SQL é referente a Tabela 2.7 e a coluna MongoDB à Figura 2.18.

Tabela 2.8. SQL e Declarações do MongoDB.

SQL	MongoDB
CREATE TABLE publicacoes (id INT NOT NULL AUTO_INCREMENT, autor Varchar(255), artigo Varchar(255), PRIMARY KEY (id))	db.createCollection("publicacoes")

ALTER TABLE publicacoes ADD data DATETIME	db.publicacoes.updateMany({ }, { $set: { data: new Date() } })
ALTER TABLE publicacoes DROP COLUMN data	db.publicacoes.updateMany({ }, { $unset: { "data": "" } })
DROP TABLE publicacoes	db.publicacoes.drop()
SELECT artigo FROM publicacoes	db.publicacoes.find({ }, { artigo: 1, autor: 0, _id: 0 })
SELECT * FROM publicacoes WHERE autor = "Stonebraker "	db.publicacoes.find({ autor: " Stonebraker " })
SELECT COUNT(*) FROM publicacoes	db.publicacoes.count()

As declarações apresentadas na Tabela 2.8 são manipulações básicas das coleções e dos documentos. Na seção 2.6 apresentam-se os operadores do método 'find', além dos métodos de agregação, agrupamento, ordenação, distinção e o uso de índices.

Na próxima seção aborda-se algumas considerações adicionais sobre SGBDs orientados a documentos, especificando como uma aplicação Java OO pode operar sobre documentos BSON.

2.5.5. Outras Considerações sobre Banco de Dados Orientados a Documentos

Os bancos de dados orientados a documentos são apenas um tipo de SGBD NoSQL e possuem vantagens e desvantagens, conforme apresentado de acordo com o teorema CAP.

Os documentos BSON criados a partir do 'BasicDBObject' podem ser criados também via objetos 'Map' do Java, antes de salvos convertidos. Tais objetos são mapeados onde cada chave possui seu respectivo valor, ou seja, através da chave é possível acessar o valor especificado, sendo que a chave não pode repetir-se, ao contrário do valor.

Considerando uma implementação Java OO (Orientada a Objeto) para o exemplo da Tabela 2.7, ao realizar a construção do objeto 'Publicações', representado na Figura 2.19, os valores são transformados em 'Map' (Figura 2.20) que por sua vez são passados para 'BasicDBObject' antes do salvamento (Figura 2.21). Um processo inverso ocorre ao recuperar os dados do banco em BSON e transforma-los em objetos Java (Figura 2.22).

```
List autor = new ArrayList();
autor.add("Tassio");
autor.add("Humberto");
Publicacoes pub = new Publicacoes("Using Ontology and Data Provenance"
        + " to Improve Software Processes", autor);
```

Figura 2.19. Instanciação de um objeto 'Publicacoes'.

```
public Map converterMap(Publicacoes pub) {
    Map mapPublicacao = new HashMap();
    mapPublicacao.put("artigo", pub.getArtigo());
    mapPublicacao.put("autor", pub.getAutor());
    return mapPublicacao;
}
```

Figura 2.20. Transformação do objeto em Map.

```
public void save(Map value) {
    BasicDBObject document = new BasicDBObject(value);
    dbCollection.save(document);
}
```

Figura 2.21. Transformação do Map para BasicDBObject.

```
public Publicacoes converterPublicacoes(DBObject dbo) {
    Publicacoes pub = new Publicacoes();
    pub.setId((ObjectId) dbo.get("_id"));
    pub.setAutor((List) dbo.get("autor"));
    pub.setArtigo(dbo.get("artigo").toString());
    return pub;
}
```

Figura 2.22. Transformação BSON para objeto.

Em Gosling *et al*. (2014) podem ser encontrados detalhes sobre orientação à objeto em Java e a manipulação de objeto Map. Esse processo de transformação de classes OO para o MongoDB e seu retorno, é uma solução para sistema já existente e que demanda a migração dos dados para um SGBD NoSQL. Um exemplo completo dessa implementação e do CRUD apresentado na seção anterior estão disponíveis no GITHUB no endereço <https://github.com/tassioferenzini/JAI >.

2.6. Operadores

O NoSQL ganha espaço no trabalho com dados não estruturados, oferecendo condições para trabalhar com grande volume de dados, de forma eficiente. Contudo, vale enfatizar que muitas das vezes são necessárias a realização de operações que exigem maior grau de conhecimento do administrador de banco de dados e as buscas, apenas pela chave, não são suficientes.

Nesse ponto, o administrador do banco deve trabalhar com operadores de comparação, resultados distintos (*distinct*), expressões regulares, operadores lógicos e unários, além de operações com texto, onde tais operadores são passados ao método 'find', o qual é responsável pela busca nos documentos. Alguns dos operadores mais comuns para manipulação de documentos BSON são demonstrados na Tabela 2.9.

Tabela 2.9. Operadores para manipulação de documentos BSON.

Operador	Explicação
$eq	Retorna os documentos que possuem valores iguais a um valor especificado. (=)

$gt	Retorna os documentos que possuem valores que são maiores que um valor especificado. (>)
$gte	Retorna os documentos que possuem valores que são maiores ou iguais que valor especificado. (>=)
$in	Retorna todo os documentos que possuem qualquer um dos valores especificados em um *array*.
$lt	Corresponde a valores que são menores que um valor especificado.
$lte	Corresponde a valores que são menores ou iguais que um valor especificado.
$and	Junta cláusulas de consulta com um AND lógico e retorna todos os documentos que correspondem às condições de ambas as cláusulas.
$not	Inverte o efeito de uma expressão de consulta e retorna os documentos que não correspondem à expressão de consulta.
$nor	Junta-se a cláusulas de consulta com um NOR lógico e retorna todos os documentos que não correspondem a ambas as cláusulas.
$or	Junta cláusulas de consulta a um OU lógico e retorna todos os documentos que correspondem às condições de qualquer uma das cláusulas.
$exists	Retorna documentos que possuem o campo especificado.

Além do método 'find', esses operadores também podem ser utilizados em métodos para documentos distintos ou agrupamentos, os quais serão apresentados na próxima seção, junto com o uso destes. Além dos operadores supracitados estão disponíveis outros para uso e podem ser encontrados com suas descrições em <https://docs.mongodb.com/manual/reference/operator/>.

2.6.1. Manipulações Avançadas

Como explicado anteriormente, o objetivo desta seção é apresentar os operadores e como podemos realizar buscas avançadas do NoSQL, onde também far-se-á um comparativo com o SQL tradicional.

Buscas avançadas são comuns em todos os tipos de aplicações, seja por meio de operadores lógicos ou matemáticos, ou para manipulação de texto, como o tradicional "*like*" do SQL. O MongoDB possui alguns métodos específicos para consultas avançadas, onde tais métodos podem ser vinculados a outros para a busca, distinção, agrupamento, agregação ou ordenação de documento, conforme já mencionado. Na Tabela 2.10 apresentam-se exemplos desses métodos e será feita uma associação com o SQL.

Tabela 2.10. Consultas SQL e os respectivos métodos correspondentes para o MongoDB.

SQL	MongoDB
SELECT * FROM publicacoes WHERE autor like "Ta%"	db.publicacoes.find({ autor : /^Ta/ })

SELECT * FROM publicacoes WHERE autor = "Tassio" ORDER BY artigo ASC	db.publicacoes.find({ autor: "Tassio" }).sort({ artigo: 1 })
SELECT * FROM publicacoes LIMIT 1	db.publicacoes.find().limit(1) ou db.publicacoes.findOne()
SELECT * FROM publicacoes LIMIT 2 SKIP 1	db.publicacoes.find().limit(2).skip(1)
SELECT DISTINCT(autor) FROM publicacoes	db.publicacoes.distinct("autor")
SELECT autor FROM publicacoes GROUP BY autor	db.publicacoes.aggregate([{ $group : { _id : { autor: "$autor"}}}])
SELECT * FROM pessoas pes INNER JOIN publicacoes pub ON pes.nome=pub.autor	db.Pessoa.aggregate([{ $lookup: { from: "Publicacoes", localField: "nome", foreignField: "autor", as: "Artigos" } }])

Um método que vale destaque é o ‘aggregate’, que serve tanto para agrupar documentos por um campo especifico, como para relacionar documentos de coleções diferentes. Essa agregação entre coleções trabalha de forma similar ao JOIN do SQL, onde os documentos agregados passam a compor um novo campo no BSON retornado pelo SGBD. Na Figura 2.23 temos uma agregação entre as coleções “Pessoa” e “Publicacoes”, onde a chave ‘nome’ da tabela “Pessoa” se relaciona com a chave ‘autor’ da tabela “Publicacoes”. Para cada documento de entrada, o ‘$ lookup’ adiciona um novo campo de matriz cujos elementos são os documentos correspondentes da coleção agregada. O resultado dessa agregação pode ser visto na Figura 2.24.

O método ‘aggregate’ também possui como opções o ‘$match’ e o ‘$project’, onde o ‘$match’ filtra os documentos para retornar somente os documentos que correspondem à condição definida, enquanto o ‘$project’’ retorna os documentos com os campos solicitados, sendo que estes campos podem existir nos documentos ou serem criados na agregação, conforme pode-se visualizar na Figura 2.25.

```
db.Pessoa.aggregate([{
    $lookup: {
    from: "Publicacoes",
    localField: "nome",
    foreignField: "autor",
    as: "Artigos"
    }

}])
```

Figura 2.23. Agregação entre coleções.

```
    "_id" : ObjectId("5ad5efaf90e8f81d50f67cfd"),
    "nome" : "Tassio",
    "sobrenome" : "Sirqueira",
    "email" : "tassio@tassio.eti.br",
    "telefone" : {
        "casa" : NumberLong("3232730000"),
        "celular" : 984440000
    },
    "endereco" : {
        "tipoLogradouro" : "Rua",
        "logradouro" : "A",
        "numero" : "1",
        "bairro" : "Centro",
        "cidade" : "Matias Barbosa",
        "cep" : "36120-000"
    },
    "Artigos" : [
        {
            "_id" : ObjectId("5ad5efaf90e8f81d50f67d00"),
            "artigo" : "Using Ontology and Data Provenance to Improve Software Processes",
            "autor" : [
                "Tassio",
                "Humberto"
            ]
        },
        {
            "_id" : ObjectId("5ad5efaf90e8f81d50f67d01"),
            "artigo" : "A Software Framework for Data Provenance",
            "autor" : "Tassio"
        }
    ]
},
```

Figura 2.24. Resultado da agregação entre coleções.

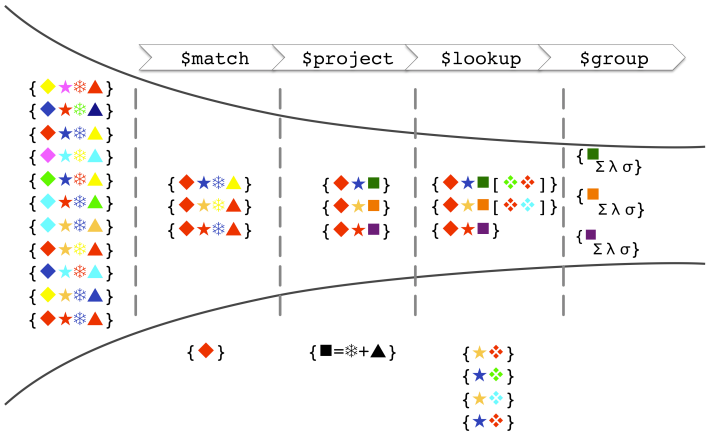

Figura 2.25. Detalhes do método 'aggregate'.

Além dos métodos apresentados, o MongoDB possui duas ferramentas para exportação e importação de dados para o SGBD, sendo estas conhecidas como 'mongoexport' e 'mongoimport', respectivamente.

O 'mongoexport' é um utilitário do SGBD que permite exportar os dados armazenados em uma instância do MongoDB, em JSON ou CSV. Já o utilitário 'mongoimport' faz o papel inverso, importando o conteúdo de arquivos JSON ou CSV para o SGBD. As figuras 2.26 e 2.27 representam respectivamente os processos de exportação e importação de uma coleção do SGBD.

```
mongoexport --db JAI --collection Pessoa --out pessoa.json;
```

Figura 2.26. Exportação de dados no MongoDB.

```
mongoimport --db JAI --collection Pessoa --file pessoa.json
```

Figura 2.27. Importação de dados no MongoDB.

2.6.2. Otimização

Quando trabalha-se com banco de dados relacionais o uso de índices são frequentes, principalmente em tabelas com grande quantidade de informações. Em bancos NoSQL o uso de índices não é diferente.

A indexação de um banco permite otimiza-lo, apresentando as consultas de forma mais rápidas e reduzindo a carga de processamento do servidor, além de garantir que uma chave não esteja duplicada. Contudo, cuidados devem ser adotados pois, pode haver o aumento do tempo de embutir no banco, aumento do consumo de memória para armazenamento em disco e para manipulação. O administrador do banco deve preocupar-se em manter o banco em estado consistente.

Os índices suportam a execução eficiente de consultas, ou seja, em uma coleção de dados sem índices é necessário verificar todos os documentos, para selecionar os documentos que correspondem à instrução de consulta. Os índices são estruturas de dados especiais que armazenam uma pequena parte do conjunto de dados da coleção em um formato fácil de percorrer.

No MongoDB o índice armazena o valor de um campo específico ou conjunto de campos, ordenado pelo valor do campo. A ordenação das entradas de índice suporta correspondências de igualdades eficientes e operações de consulta baseadas em intervalo. Além disso, o MongoDB pode retornar resultados classificados usando a ordenação no índice.

O SGBD oferece alguns métodos para manipulação de índices, conforme pode-se observar na Tabela 2.11, que podem operar sobre uma única chave ou múltiplas chaves.

Tabela 2.11. Manipulação de índices no MongoDB.

Método	Descrição
createIndex()	Cria um índice em uma coleção.
createIndexes()	Cria um ou mais índices em uma coleção.
reIndex()	Recria todos os índices existentes em uma coleção.
getIndexes()	Retorna um vetor que descrevem os índices existentes em uma coleção.
dropIndex()	Remove um índice específico em uma coleção.
dropIndexes()	Remove todos os índices de uma coleção.

Conforme mencionado, a criação de índices pode ser sobre uma chave específica dos documentos de uma coleção ou sobre múltiplas chaves, conforme apresenta as figuras 2.28 e 2.29 respectivamente, onde o valor ‘1’ especifica que o índice será ascendente e o ‘-1’ descendente.

```
db.getCollection("Pessoa").createIndex({ "endereco.cep": 1 })
```

Figura 2.28. Importação de dados no MongoDB.

```
db.getCollection("Pessoa").createIndex({ "endereco.cep": 1, "endereco.numero": -1 })
```

Figura 2.29. Importação de dados no MongoDB.

Um índice no MongoDB pode ser geoespacial, texto, único, parcial ou esparso, onde detalhes podem ser consultados na documentação[25]. Além dos índices, o SGBD oferece o método 'mapReduce'. O Map-reduce é um paradigma de processamento de dados para condensar grandes volumes de dados em resultados agregados, conforme exibe-se na Figura 2.30.

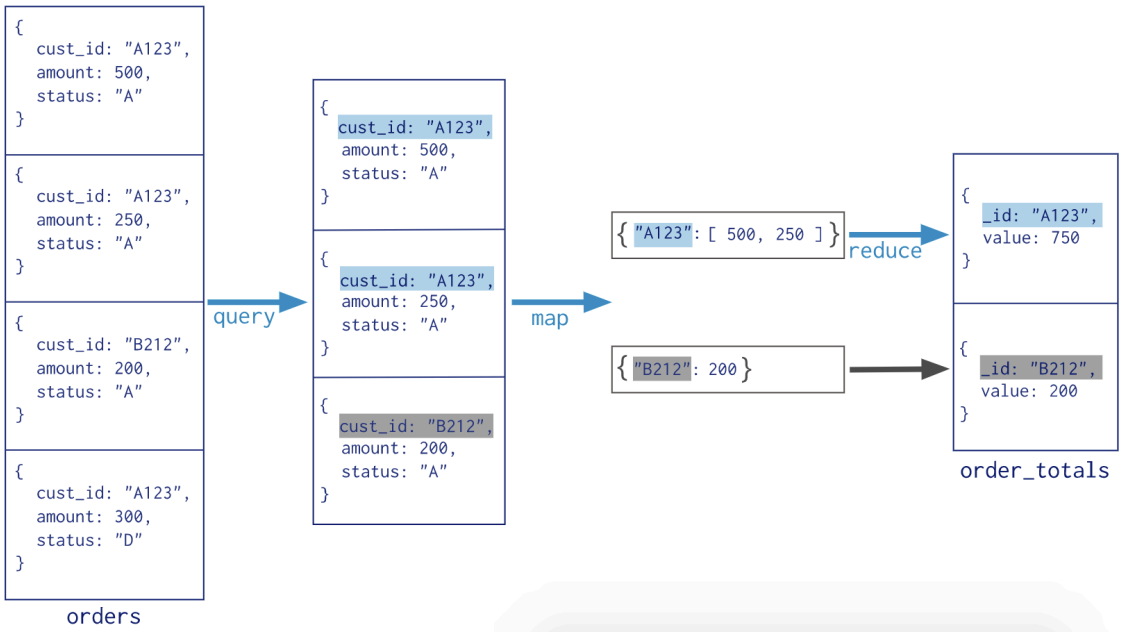

Figura 2.30. MapReduce no MongoDB.

Para a execução de um 'mapReduce', deve-se criar uma função Map, responsável por mapear os objetos que serão agrupados, e uma função de redução, a qual definirá o resultado esperado pela redução. Um exemplo do método 'mapReduce' para a tabela 2.1 retornando as cidades e quantas vezes elas aparecem na coleção 'Pessoa' é apresentado na Figura 2.31.

```
var map = function(){
  emit(this.endereco.cidade, 1);
}
var reduce = function(key, values){
  var res = 0;
  values.forEach(function(v){ res += 1});
  return {count: res};
}
db.Pessoa.mapReduce(map, reduce, { out: "map_cidade" });
```

Figura 2.31. Exemplo de MapReduce no MongoDB.

[25] Índices no MongoDB: Disponível em <https://docs.mongodb.com/manual/indexes/ - index-types>.

2.7. O Futuro do NoSQL

Os bancos NoSQL estão mais evoluídos e a "guerra fria" entre NoSQL vs. SQL está ficando antepassada. Atualmente aplicações tendem a ser "poliglotas" e buscar o melhor, vindo a utilizar a respectiva tecnologia de banco de dados que adequa-se a necessidade da aplicação.

Os bancos criados utilizando NoSQL não foram desenvolvidos em substituição aos bancos relacionais, haja vista que os mesmos trabalham com dados em que os relacionais tendem a apresentar gargalos. Todavia os bancos NoSQL precisam amadurecer em alguns quesitos, já sendo notórias algumas contribuições.

Atualmente os SGBDs NoSQL estão ganhando frente entre os produtos de bancos de dados mais usados mundialmente, o que demonstra que o mesmo vem sendo adotado em massa por especialistas em TI e, portanto, pode ser também uma boa opção para o projeto sendo desenvolvido atualmente.

Segundo a pesquisa desenvolvida em 2017 pelo Stack Overflow[26], de 29452 usuários que responderam sobre qual SGBD tem utilizado em seus projetos, 21% responderam o MongoDB, sendo esse o SGBD o mais bem posicionado na pesquisa entre os SGBD NoSQL, ocupando a 5ª colocação.

Nessa mesma pesquisa, o MongoDB ocupou a 3ª colocação entre os SGBD mais amados pelos desenvolvedores, atrás do Redis (SGBD NoSQL) e do PostgreSQL. Além disso, foi apontado como o SGBD mais desejado pelos desenvolvedores, conforme Figura 2.32.

Na pesquisa do ano anterior feita pelo Stack Overflow, o Redis era o banco de dados mais amado, o que significa que, proporcionalmente, mais desenvolvedores queriam continuar trabalhando com ele do que qualquer outro banco de dados. Com o apontamento dos desenvolvedores em indicar o MongoDB em 2017 como o mais desejado, caracteriza que para este ano de 2018 esses resultados mudem.

O ponto principal desta pesquisa é o avanço dos SGBDs NoSQL, que até então não despontavam. Isso representa uma quebra de paradigmas, é a mudança de visão dos profissionais de TI em busca de novas soluções no campo da engenharia do conhecimento.

[26] Pesquisa Stack Overflow: Disponível em <https://insights.stackoverflow.com/survey/2017>.

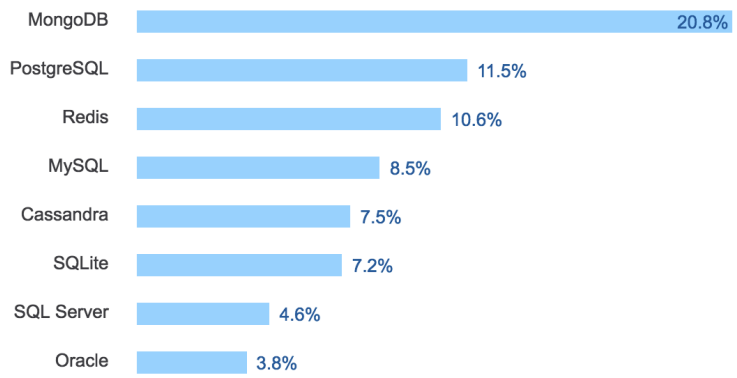

Figura 2.32. SGBD mais desejado pelos desenvolvedores.

Cabe ressaltar que SGBDs NoSQL não são para substituir os SGBDs relacionais, conforme já citado, e também não é a solução de todos os problemas. As características do NoSQL tornam-no atrativo na resolução de alguns problemas. Na próxima seção abordaremos a aplicabilidade dos SGBD NoSQL e quando essa tecnologia de banco de dados deve ser aplicada.

2.7.1. Aplicabilidade do NoSQL nos problemas do Mundo Real

Atualmente diversas áreas já abordam grandes volumes de dados, podendo citar-se como exemplo empresas do ramo financeiro, logística, saúde e redes sociais. Várias empresas incorporadas a estes ramos já vêm investindo em banco de dados NoSQL, sendo essa necessidade decorrente de grande volume de dados, falta de estrutura nos dados, descentralidade ou estabilidade dos dados persistidos.

Atualmente, o volume de dados de certas organizações, vem expandindo-se rapidamente, onde a utilização dos SGBDs relacionais tem mostrado-se muito problemática e não tão eficiente.

Dentre os principais problemas na utilização do Modelo Relacional estão a dificuldade de conciliar o tipo de modelo com a demanda da escalabilidade, a qual é cada vez mais frequente. Afim de exemplificar, suponhamos que o sistema está trabalhando sobre um SGBD relacional, onde houve um crescimento de usuários, o que culmina em uma queda de performance. Para resolver este problema, tende-se a fazer um upgrade no servidor ou aumentar o número destes.

A solução apresentada só funcionaria caso o número de usuários estagnasse ou crescesse vagarosamente, haja vista que o problema passa a se concentrar no acesso à base de dados. Neste caso, solucionar-se-ia o problema aumentando o poder do servidor, mediante ao aumento da memória, processador e armazenamento. Esta solução é chamada de Escalabilidade Vertical. Também poder-se-ia aumentar o número de máquinas no servidor web, onde esta alternativa é denominada Escalabilidade Horizontal, conforme explicado no teorema CAP.

Ao observar-se o intenso volume de dados e sua natureza estrutural, os desenvolvedores notam a dificuldade de se organizar os dados no Modelo Relacional sobre sistemas distribuídos. Neste ponto, o foco das soluções não-relacionadas está direcionado.

No intuito de solucionar diversos problemas relacionados à escalabilidade, performance e disponibilidade, projetistas do NoSQL trabalharam em uma alternativa de alto armazenamento, com velocidade e grande disponibilidade, abstraindo certas regras e estruturas que norteiam o Modelo Relacional.

A proposta dos bancos NoSQL não visa extinguir o Modelo Relacional, mas sim utilizá-lo em casos onde é necessária uma maior flexibilidade na estruturação do banco e melhor desempenho. Podemos utilizar um banco NoSQL e um relacional em conjunto, onde o SGBD NoSQL é indicado para que sistema que necessitem de alta disponibilidade ou escalabilidade e os dados que não necessitam dessa abordagem, podem ir para uma base relacional. Dessa forma temos uma utilização muito mais eficiente e econômica dos recursos.

Essa abordagem de um SGBD NoSQL foi utilizada com sucesso em uma plataforma de análise de dados econômicos, onde são incorporados ao sistema aproximadamente 27 milhões de documentos (registros) por mês, fazendo que o armazenamento, a escalabilidade e o desempenho levassem a plataforma a depender de uma solução alternativa aos clássicos SGBDs relacionais.

Esse mesmo problema vêm sendo enfrentado por uma aplicação médica, onde dados são coletados e apresentados em tempo real, e devem ser armazenados para histórico médico. Nesse caso, o principal problema é a estrutura dos dados, pois são provenientes de um equipamento com sensores ligados ao paciente, e cada equipamento transmite os dados conforme os sensores que estão ligados ao paciente, tornando a composição dos dados dinâmica.

Algumas empresas como Google, Facebook, GitHub e Globo já vem utilizando soluções NoSQL em seus produtos e como apontado pelo Stack Overflow, o crescimento e adoção desses SGBDs NoSQL são necessários em diversos cenários para solução de problemas computacionais.

Atualmente são os principais SGBD quando o assunto é *Big Data*, o que obrigas os profissionais da área a se atualizarem para esta nova tecnologia. Manipular dados considerados do *Big Data* já dissemina uma área própria dentro da engenharia de dados, chamada de '*Data Science*'.

Conforme abordado por Matos (2016), a Ciência de Dados (*Data Science*) é o domínio científico dedicado à descoberta de conhecimento através da análise de dados, onde Cientistas de Dados utilizam técnicas matemáticas e algoritmos para encontrar soluções de problemas de negócio ou científico.

Engenheiros de Dados que não atentarem as essas mudanças se limitarão a problemas triviais. O propósito do Engenheiro de Dados, é fornecer soluções que possam enriquecer os dados por ele gerenciado.

Na próxima seção abordaremos o perfil dos novos profissionais que atuarão nesta área.

2.7.2. O Novo Perfil dos Profissionais da Computação

Conforme supracitado, o NoSQL representa uma mudança de paradigma e, atualmente, grande parte dos cursos de computação ainda seguem a tradição de apresentarem apenas bancos de dados relacionais, sua modelagem e o padrão SQL.

Mediante ao crescimento do NoSQL é necessário compreender essa filosofia, frente as novas demandas do mercado e a necessidade de profissionais que possuam esse novo conhecimento. Esse novo perfil de profissional, o qual conhece banco de dados NoSQL, implica na atualização das grades dos cursos de computação e na atualização de centenas à milhares de profissionais que já atuam com banco de dados relacional.

Atentando mais especificamente em Engenharia de Software, a qual encontra-se na área de Software e Serviços, a descrição do profissional desejado compreende especificações entre a visão, o papel e o estilo de vida. A visão engloba criação, teste, instalação e manutenção do software, visando novos sistemas de informação para o mercado. A inteligência tecnológica em quaisquer dos ambientes de desenvolvimento e domínios de aplicação também é vital, desde que a habilidade de captar as necessidades dos clientes seja tão importante.

Já no papel vislumbra-se que o Engenheiro de Software desenhe, construa, teste, implemente e mantenha aplicações que atendam necessidades específicas do cliente. As aplicações mencionadas incluem aplicações para empresas, aplicações de software embebido, tais como para dispositivos móveis e planejamento de recursos de empresas, sistemas em ambiente de negócios e industrial. Conhecimento sobre a interação humano/computador é também parte do papel a ser englobado pelo Engenheiro de Software, o qual tende a envolver a psicologia humana, ergonomia, assim como desenvolvimento de aplicações.

Quanto ao estilo de vida, na maioria dos casos o trabalho será constituído a cabo em equipe, sendo possível que equipes trabalhem em múltiplos sítios e comuniquem-se através de dispositivos de comunicação. Inicialmente à atividade, é necessário obter características técnicas com o resto da equipe, sendo necessário, ao passar do tempo, um maior envolvimento com os negócios e ambiente dos clientes, afim de demonstrar e implementar as aplicações e soluções.

Os engenheiros de software que irão trabalhar com o sistema para o *Big Data* ou com sistemas que demandem o uso de um SGBD NoSQL, precisam conhecer essa

tecnologia para terem condições de modelar o sistema seguindo esse novo paradigma e saberem também, quando um projeto irá demandar uma solução flexível e escalável, evitando assim que projetos fracassem ou ultrapassem o orçamento por conta de reengenharia.

Já os engenheiros de dados são os responsáveis por manter essa estrutura de dados, garantindo a escalabilidade, consistência dos dados e fornecendo mecanismos para os analistas de dados extraírem informações úteis das bases, o que não é uma tarefa trivial frente ao *Big Data*.

Para que soluções envolvendo o uso de SGBDs NoSQL deem certo, é necessário além da comunicação entre os engenheiros de dados e de software, que ambos estejam alinhados com essa tecnologia, buscando extrair o melhor de cada SGBD para cada solução provida. Profissionais que não se atualizarem, ficaram limitados a sistemas triviais.

2.8. Considerações Finais

O objetivo desse minicurso é apresentar, de modo teórico e prático, o uso de NoSQL e as mudanças que envolvem a engenharia de dados e a engenharia de software para o uso dessa nova tecnologia. Na parte prática foi apresentado como modelar e construir aplicações em Java utilizando o MongoDB, buscando alinhamento da teoria à prática.

A alteração do uso de um banco de dados relacional para um banco de dados não-relacional é desafiadora, sendo necessário um estudo profundo sobre os bancos de dados não relacionais que vêm a cumprir todas as demandas de um dado usuário e/ou desenvolvedor, dado que cada solução demanda suas próprias características.

Criar um banco de dados não relacional tende a fornecer os mesmos recursos e operações de consulta abrangidos por um banco de dados relacional, sendo que em geral a performance é melhor. A utilização do banco de dados relacional não tende a ser findada, haja vista que o mesmo fornece um conjunto de recursos incomparável, tal como a integridade e a confiabilidade dos dados, motivo pelos quais são amplamente utilizados.

O NoSQL é uma tecnologia que vem divulgando-se no mercado, principalmente quando trata-se de *Big Data*. Os profissionais da área de computação devem ser capazes de trabalhar com a integração entre banco de dados relacional e banco de dados NoSQL (não relacional), em busca das melhores soluções.

O MongoDB é um banco de dados NoSQL bastante popular, assim como a linguagem de programação Java. Contudo, existem diversos SGBDs NoSQL além do MongoDB, voltados para diferentes soluções e atualmente disponíveis para uso pelas principais linguagens de programação.

Nesse minicurso o intuito é abranger um público que posteriormente possa vir a utilizar outro SGBD NoSQL, assim como outras linguagens de programação como o Python, que é bastante comum em aplicação que utiliza o SGBD NoSQL.

Ao analisar os dados imputados neste trabalho, observa-se um melhor desempenho do MongoDB e uma modelagem fora do tradicional. É importante ressaltar que isso não implica em uma superioridade total do MongoDB, porém demonstra algumas das funções atendidas por este.

Esse trabalho apresenta uma visão geral da área e da integração entre a engenharia de dados e de software, para soluções de problemas do *Big Data* ou para sistemas especializadas. Esperamos que em breve o ensino de SGBD NoSQL seja disciplina básica entre os cursos de graduação, assim como é hoje com os SGBDs relacionais, capacitando os novos profissionais da computação, para a resolução dos desafios do *Big Data*.

Referências

_____. (2018). MongoDB Docs. Disponível em: <https://docs.mongodb.com/>. Acessado em: 19 de abr. de 18.

_____. Engenharia de Software, Banco de Dados e Interação Humano Computador. Disponível em: <http://ppgcc.dc.ufscar.br/pesquisa/linhas-de-pesquisa/engenharia-de-software>. Acessado em: 26 de fev. de 18.

Amaral, F. (2016). Introdução à Ciencia de Dados: mineração de dados e big data. Alta Books Editora.

Brewer, E. A. (2000, July). Towards robust distributed systems. In PODC (Vol. 7).

Cattell, R. (2011). Scalable SQL and NoSQL data stores. Acm Sigmod Record, 39(4), 12-27.

Date, C. J. (2004). Introdução a sistemas de bancos de dados. Elsevier Brasil.

Dhar, V. (2013). Data science and prediction. Communications of the ACM, 56(12), 64-73.

Elmasri, R., Navathe, S. B., & Pinheiro, M. G. (2005). Sistemas de banco de dados.

Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2014). The Java Language Specification, Java SE 8 Edition (Java Series).

Grolinger, K., Higashino, W. A., Tiwari, A., & Capretz, M. A. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. Journal of Cloud Computing: advances, systems and applications, 2(1), 22.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, 29(7), 1645-1660.

Horowitz, E. (2018). Multi-document transactions in MongoDB. Disponível em: <https://www.mongodb.com/blog/post/multi-document-transactions-in-mongodb>. Acessado em: 15 de abr. de 18.

Leavitt, N. (2010). Will NoSQL databases live up to their promise?. Computer, 43(2).

Lopes, T., Cavaleiro, J., Rocha, J., Rodrigues, P. (2005). Perfis, Papéis e Competências em Engenharia de Software. FEUP. Universidade do Porto. Portugal.

Matos, D. (2017). Cientista de Dados x Engenheiro de Dados. Disponível em: <http://www.cienciaedados.com/cientista-de-dados-x-engenheiro-de-dados/>. Acessado em: 26 de fev. de 18.

Mayer-Schonberger, V., & Cukier, K. (2014). Big data: como extrair volume, variedade, velocidade e valor da avalanche de informação cotidiana (Vol. 1). Elsevier Brasil.

Paniz, D. (2016). NoSQL: Como armazenar os dados de uma aplicação moderna. Editora Casa do Código.

Sommerville, I. (2011). Engenharia de software. 9 ed. Pearson Prentice Hall.

Strauch, C., Sites, U. L. S., & Kriha, W. (2011). NoSQL databases. Lecture Notes, Stuttgart Media University, 20.