CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

JONATAS MILTON DA CRUZ

NOSQL - ESQUEMAS FLEXÍVEIS PARA GERENCIAMENTO DE BANCO DE DADOS

Americana, SP

2015

CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

JONATAS MILTON DA CRUZ

NOSQL - ESQUEMAS FLEXÍVEIS PARA GERENCIAMENTO DE BANCO DE DADOS

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do Prof. Me. José Mario Frasson Scafi.

Área de concentração: Bancos de dados.

Americana, S. P.

2015

Cruz, Jonatas Milton da

C962n NOSQL: esquemas flexíveis para gerenciamento de banco de dados. / Jonatas Milton da Cruz. – Americana: 2015.
55f.

Monografia (Graduação em Tecnologia em Análise e Desenvolvimento de Sistemas). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Me. José Mário Frasson Scafi

1. Banco de dados I. Scafi, José Mário Frasson II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.3.07

Jonatas Milton da Cruz

NOSQL – ESQUEMAS FLEXÍVEIS PARA GERENCIAMENTO DE BANCO DE DADOS

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo CEETEPS/Faculdade de Tecnologia – Fatec/ Americana.
Área de concentração: Banco de dados

Americana, 23 de junho de 2015.

Banca Examinadora:

José Mario Frasson Scafi (Presidente)
Mestre
Fatec Americana

Francesco Artur Perrotti (Membro)
Mestre
Fatec Americana

Rogério Nunes de Freitas (Membro)
Especialista
Fatec Americana

AGRADECIMENTOS

Em primeiro lugar gostaria de agradecer a Deus pela capacidade e ajuda durante todos esses anos. Agradeço também a minha família por todo suporte e ajuda principalmente a minha mãe Ivanir, meu irmão Anderson, minha irmã Priscila e minha namorada Keity. Meu agradecimento também vai para os meus colegas e amigos de faculdade que ajudaram que esses anos fossem mais divertidos e mais fáceis. Agradeço ao meu professor orientador José Scaffi pelo auxilio no desenvolvimento e aos demais professores da instituição pela dedicação e ensino.

DEDICATÓRIA

Dedico esse trabalho a minha família, aos professores e amigos.

RESUMO

Os dados que necessitam serem salvos cada dia são maiores e cada sistema tem sua particularidade e complexidade que precisam ser resolvidas de maneira diferente. O modelo relacional de armazenamento de dados mostrou-se satisfatório ao longo das últimas décadas por ser utilizado em quase todos os sistemas atuais possibilitando vários tipos de transações de dados, porém repensar na forma de organizar os dados e na maneira que ele seria armazenado tornou-se necessário pela quantidade de registros gerados a cada instante e que precisam ser gravados e recuperados da base de dados que estará sendo acessado por múltiplos usuários simultaneamente. Este trabalho tem como objetivo conceituar os principais tópicos do modelo relacional, como o que é um dado, uma tabela, cardinalidades e integridade dos dados, além do modelo NOSQL e suas principais características, como o que é um agregado, a repartição dos dados e sua flexibilidade na criação dos bancos de dados. Um estudo de caso nesse projeto planeja um banco de dados que utiliza o modelo não relacional de dados e a persistência poliglota para armazenar e recuperar dados. Como resultado alcançado, este estudo oferece uma nova maneira de desenvolver bancos de dados e da utilização de sistemas NOSQL em vários tipos de projeto.

Palavras Chave: Banco de dados; NOSQL; Persistência poliglota.

ABSTRACT

The data that should be saved are larger every day and each system has its own complexity and difficulty that need to be resolved differently. The relational model of data storage was satisfactory over the past decades used in almost all current systems enabling various types of data transactions, but think in another way to organize and storage the data it became necessary by the quantity of records created every moment and because this data need to be recorded and recovered by multiple users at the same time. This paper aims to conceptualize the main topics of relational model, as which is a record, a table, cardinalities and integrity of the data, beyond the NoSQL model and its main features, as which is an aggregate, the repartition and flexibility in database creation. This paper also includes a case study that plan a database using the NoSQL model and polyglot persistent to storage and recovery the data. As a result achieved, this paper offers a new way to design databases and its utilization in NoSQL DBMSs.

Keywords: *Database; NoSQL; Polyglot persistence.*

SUMÁRIO

LISTA DE FIGURAS E DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CQL	Linguagem de consulta do sistema gerenciador Cassandra
DDL	Linguagem de definição de dados
DML	Linguagem de manipulação de dados
JSON	Objeto de Notação javascript
NOSQL	Banco de dados não relacional
SGBD	Sistema gerenciador de banco de dados
SQL	Linguagem de consulta estruturada
UML	Linguagem unificada de modelagem
XML	Linguagem de marcação para necessidades especiais

1 INTRODUÇÃO

O número de pessoas com acesso a internet em 2000 era de pouco mais de quatro milhões, já no final de 2014 esse número chegou a quase três bilhões de pessoas, aproximadamente 40% do total da população mundial. [1] Muitas empresas aproveitando esse impressionante crescimento ao longo dos anos aproveitaram para entrar no mundo do comércio eletrônico, sem contar as redes sociais, que servem para unir virtualmente pessoas conhecidas, ou que tenham algum gosto em comum. Todos esses fatores contribuíram para o aumento dos dados que são transferidos pela internet, e a espera pode causar desconforto em usuários.

O termo *NOSQL* foi criado para referenciar uma conferência entre entusiastas de bancos de dados não relacionais, que gostariam de arrumar um nome curto que funcionasse como um *hashtag* do *Twitter,* fosse fácil de decorar e conseguisse transmitir uma ideia ainda que superficialmente. (Sadalage *et al,* 2013).

O *NOSQL* existe pela necessidade de uma manipulação maior de dados e ganhou importância também por uma discussão antiga do modelo relacional e algumas de suas dificuldades. Sendo uma delas a diferença no modo em que os dados são armazenados na memória e como são modelados no conceito relacional. (Sadalage *et al,* 2013).

Para Sadalage *et al* (2013), o conhecimento de uma nova tecnologia pode levar os programadores a uma persistência poliglota, ou seja, usar mais um modelo de banco de dados para persistir os dados de um mesmo sistema, e escolher diferentes maneiras de gerenciar o seu banco de dados, adaptando-o a necessidade do seu projeto. Mas o surgimento e o crescimento dos bancos de dados *NOSQL* não implicam que o modelo relacional só tem defeitos, haja vista que as principais qualidades, como: persistir grande quantidade de dados, a capacidade de lidar com a concorrência, ter uma estrutura padronizada e possibilitar a integração de vários aplicativos acessando a mesma base de dados, ainda possam pesar a seu favor na hora de escolher o modelo de banco de dados do seu projeto.

Sistemas de bancos de dados *NOSQL* cresceram e ganharam adeptos principalmente pela necessidade de às vezes fugir da estrutura relacional, como por exemplo, a criação de dados aninhados, ou a utilização de *clusters* no projeto, visto

[1] Internet Users, Internet Users in the World. Disponível em: <http://www.internetlivestats.com/internet-users/> Acesso em 10 de maio de 2015.

que o modelo relacional não foi desenvolvido para ser usado dessa maneira. Ao encontrar essas barreiras grandes companhias começaram a pensar em novas soluções e o tema ganhou força, quando no início da década 2000, o *Google* e a *Amazon* lançaram seus bancos de dado não relacionais.

O estudo se **justifica** pela crescente quantidade de tráfegos de dados, aumento no volume de transações pela internet e também no aumento de dispositivos capazes de acesso a informações. Além da importância de conhecer novas tecnologias de gerenciamento de bancos.

O problema foi: o crescente aumento de usuários em determinados sistemas, aumentando constantemente o número de dados, levando as empresas a gastarem mais com aluguel de servidores externos ou então aumentando sua infraestrutura.

A pergunta que se buscou responder: o banco de dados denominado *NOSQL* poderá ganhar espaço nas empresas na hora do desenvolvimento?

As **hipóteses** foram: a) Essa metodologia tornou-se inviável, visto que é de difícil aplicação e a sua manutenção muito trabalhosa e custosa. b) A metodologia mostrou-se prática, podendo aperfeiçoar o processo de manipulação de dados, porém a mão de obra precisa ser qualificada, podendo tornar o método inviável. c) A metodologia atende todas as expectativas, sua implantação e manutenção podem ser facilmente compreendidas e seu resultado de otimização foi considerado satisfatório.

O **objetivo geral** constituiu pesquisar, analisar e exibir as novas tendências existentes no mercado para gerenciamento de banco de dados, bem como sua utilização adequada. O estudo buscará auxiliar futuros projetos na escolha de uma melhor maneira de organizar a estrutura de armazenamento dos dados.

Os **objetivos específicos** foram: a) Pesquisar sobre o modelo relacional e as suas características, bem como vantagens e desvantagens. b) Pesquisar e levantar dados sobre gerenciamento e estruturação de banco de dados utilizando os conceitos de *NOSQL*. c) Exibir exemplos práticos, e seus resultados, de banco de dados seguindo a estruturação e os dados levantados no item b, além de comparação com o modelo de gerenciamento de dados pesquisado no item a. d) Discutir os resultados obtidos nos itens a, b e c, objetivando a chegar a conclusões sobre os mesmos.

O **método de pesquisa** utilizado para o desenvolvimento desse trabalho, do ponto de vista da sua natureza, foi: a pesquisa aplicada, que de acordo com Gil (2002), objetiva gerar conhecimentos para aplicação prática dirigida à solução de problemas específicos.

Do ponto de vista de seus objetivos, a pesquisa é exploratória, visto que visa proporcionar familiaridade com o problema, tornando-o explicito ou na tentativa de criar hipóteses. Já do ponto de vista dos procedimentos técnicos esse estudo pode ser enquadrado como estudo de caso, que é quando, segundo Gil (2002), envolve um estudo profundo e exaustivo de um ou poucos objetos, de maneira que possa permitir um detalhado conhecimento sobre o assunto. Utiliza-se essa pesquisa para expandir o conhecimento em sistema gerenciador de banco de dados e melhorar a tomada de decisão antes do início de um novo projeto.

Quanto à **organização do estudo**, no primeiro capítulo vemos uma introdução que apresenta o tema proposto, bem como a sua justificativa, o problema, as hipóteses, o objetivo geral, os objetivos específicos e o método de pesquisa. No segundo capítulo define-se e conceitua-se sobre o que é um banco de dados e a importância de armazenar dados. É também visto o modelo relacional e seus principais conceitos, como relacionamentos, entidades, atributos, tuplas, transações, a integridade, a linguagem *SQL*, e o que é e a importância dos SGBDs. O terceiro capítulo conceitua o que é *NOSQL*, porque se fez necessário, bem como suas principais características e como ele atuando quando trabalhando em clusters de computadores. Nesse capítulo também conceitua-se os principais modelos de bancos *NOSQL* e a chegada à persistência poliglota. No quarto capítulo, um estudo de caso é proposto sobre um serviço de vendas pela internet, utilizando os conceitos vistos no capítulo três, mostrando como a persistência poliglota funcionaria na prática. No capítulo cinco são descritas as conclusões que esse trabalho chegou.

2 O BANCO DE DADOS E O MODELO RELACIONAL

Um dado é uma representação simbólica e que pode ser quantificada, como uma palavra ou uma imagem que também pode ser lida como uma sequência de símbolos quando digitalizada. Uma foto de um objeto se torna um dado, porém o objeto real não, visto que um objeto real não é só o que pode ser visto, mas também contém atributos como peso, altura, profundidade, cor, tempo de vida e etc., esses atributos de um objeto real sim podem ser quantificados e qualificados como dados. (Setzer *et al*, 2005).

Ressalta-se ainda que informações são mensagens que foram enviadas como dados, porém só se tornam informações se o receptor for capaz de associar algum significado a esse conjunto de dados, senão serão somente símbolos sem significados. O computador não transmite informações, ele simplesmente processa dados que pode representar informações, como por exemplo, ao olhar uma tabela contendo nome e data de nascimento se pode chegar à informação sobre quem é o mais velho baseado nos dados que o computador exibiu. Sendo assim, o sentido de uma base de dados é armazenar dados, que em um determinado momento serão requisitados e exibidos aos usuários, podendo então ser transformados em informações. (Setzer *et al*, 2005).

Um banco de dados é o local onde são armazenados registros, equivalente a um armário de escritório onde é comum armazenar registros. Onde é possível inserir, atualizar, buscar e remover os dados. Um banco de dados também pode ser definido como um conjunto de dados organizados logicamente. (Date, 2004).

Bancos de dados estão presentes em quase todos os sistemas e em aparelhos que podem variar de celulares, mp4, computadores de uso doméstico até aos maiores mainframes, e são indispensáveis hoje em dia, por exemplo, um banco armazena informações bancárias de uma pessoa, sendo capaz de manter a salvo as últimas transações realizadas.

Ainda segundo Date (2004), bancos de dados podem ser uma coleção de dados persistentes, onde por persistente pode-se entender que após esses registros serem inseridos através de um SGBD (Sistema Gerenciador de Banco de Dados) em um base de dados, só poderão ser removido por uma ação do SGBD.

Um banco de dados se faz necessário sempre que houver a necessidade de armazenar grandes quantidades de informações. Em caso de sistemas de pequeno porte, de um único usuário, por exemplo, talvez as vantagens de utilizar um banco de dados diminuam, mas em caso contrário, com uma quantidade maior de pessoas tendo necessidade de acessar a informação e/ou que essas informações sejam em grandes quantidades, as vantagens de utilizar banco de dados entre outras são: densidade (Não há volume como se fosse feito em papel), velocidade (consulta, atualizações, inserções e exclusão podem ser feitas de maneira rápidas.), atualidade (dados mais precisos e com disponibilidade para mais pessoas acessarem) e proteção. (Date, 2004).

Entre outras coisas, um banco de dados também é importante pela capacidade dos dados serem compartilhados, ou seja, várias aplicações podem acessar os seus dados e em um banco de dados também se pode diminuir a duplicidade de registros. (Date, 2004).

O total de registros que um banco de dados é capaz de armazenar está diretamente ligado à capacidade do hardware onde ele está instalado, ou do cluster de computadores. É também o hardware da máquina responsável por influenciar no desempenho dos sistemas gerenciadores de banco de dados (SGBD).

2.1 ENTIDADES

Um banco de dados (modelo relacional) é basicamente dividido em tabelas, linhas onde ficam os registros do arquivo e as colunas que são os campos, mas esses termos também são conhecidos como relações, tuplas e atributos. (Date, 2004).

Como observado na **tabela 1**, onde é demonstrada uma tabela denominada Funcionários que contém como atributos ID, Nome, Cargo e Admissão.

TABELA 1 - Exemplo de uma tabela

Tabela de Funcionário			
ID	**Nome**	**Cargo**	**Admissão**
1	**José da Silva**	**Zelador**	**12/04/19747**
2	**Maria Santos**	**Gerente**	**08/09/1995**
3	**Zélia Santos**	**Produção**	**01/10/2002**

Fonte: Autoria própria

Uma entidade deve representar objetos reais de mesma categoria, portanto um conjunto de pessoas, um conjunto de cargos e um conjunto de livros devem ser representados como três entidades diferentes. Entidades contêm atributos que são as informações que se deseja armazenar e saber sobre determinado item, como por exemplo, pessoas podem conter atributos como nome, sexo e CPF. (Setzer *et al,* 2005).

O modelo entidade-relacional (MER) é uma forma de representar o relacionamento entre entidades, onde uma entidade fica localizada em um retângulo, e seus atributos são representados com um traço e uma circunferência em uma das extremidades. (Setzer *et al*, 2005).

Entidades podem ser classificas como fortes: em que sua existência não depende de outras entidades, por exemplo, em um sistema de biblioteca, a entidade pessoa não necessita de nenhuma outra para existir. Entidades fracas: diferentemente das fortes, as entidade fracas precisam que outras entidades existam para existir, como uma entidade que armazenaria o telefone das pessoas, não faria sentido se não houvesse a entidade de pessoas. Por fim existem também as entidades associativas que surgem quando há um relacionamento de muitos para muitos entre entidades.

Sobre entidades, Oppel (2009) diz que pode ser uma pessoa, um lugar, um evento, uma coisa ou um conceito sobre quais se tem interesse para armazenar dados sobre eles em um banco de dados e são interligados por relacionamentos, que são a cola que mantém a base de dados unida.

A cardinalidade dos relacionamentos representa o número mínimo e o número máximo de instâncias que uma determinada entidade pode associar com a outra entidade que está relacionada. O número mínimo de cardinalidade entre duas entidade é zero, e o número máximo pode variar entre um e muitos, dependendo da situação. (Oppel, 2009).

Em um cardinalidade *1-1*, uma instância, ou um registro, de uma entidade pode se relacionar no máximo com uma instância da outra entidade e vice versa. Supondo que exista uma entidade para Prefeitos e uma entidade para Cidades, portanto um prefeito estaria ligado a apenas uma cidade, e uma determinada cidade poderia estar relacionada a apenas um prefeito. Relacionamento *1-1* são raros em

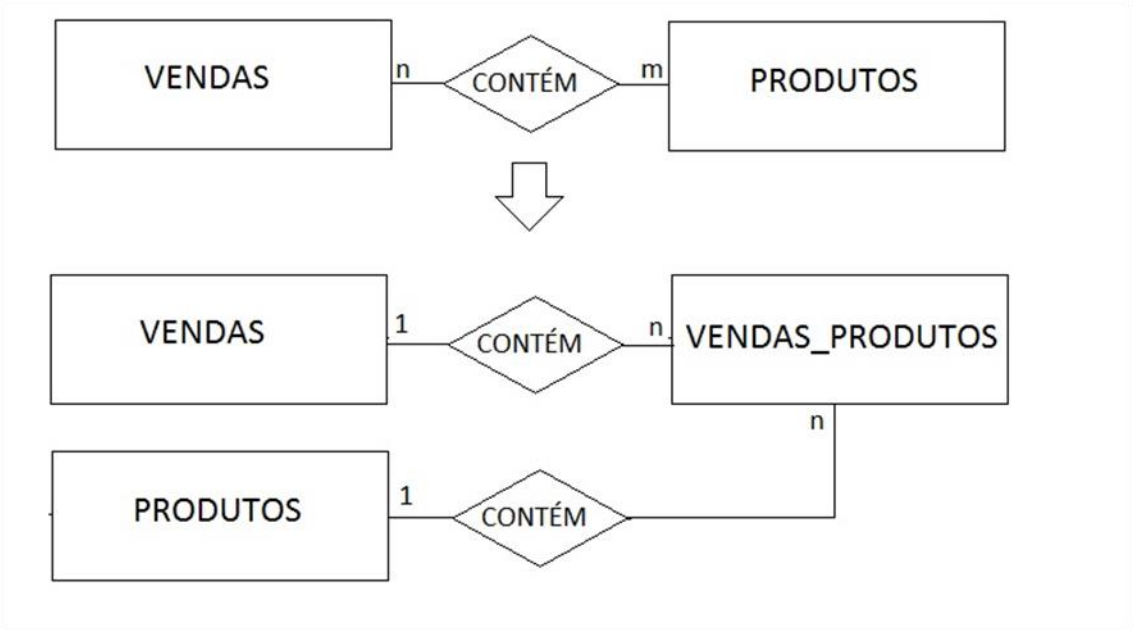
bancos de dados, visto que em algumas dessas situações, a entidade fraca pode ser absolvida pela entidade forte. (Oppel, 2009).

Na Cardinalidade *1-n* (uma para muitos), uma das entidades pode se referenciar há varias instâncias da outra, mas essa segunda entidade só pode ser referenciada a uma instância da primeira, como no exemplo de pessoas e telefones, supondo que uma pessoa pode conter vários telefones, porém cada telefone seria referenciado para somente uma pessoa. Existe também uma variação dessa cardinalidade, que seria *0-n*, que implicaria que, no exemplo proposto, um cliente poderia ter no mínimo 0 e máximo n instâncias de telefones. (Oppel, 2009).

Em uma cardinalidade *n-m* (muitos para muitos) uma entidade pode se referenciar a muitas instâncias de outra entidade e vice-versa, um exemplo muito comum disso é com entidades de vendas, em que uma entidade com o nome Vendas e uma entidade Produtos precisam ser relacionados, porém com um relacionamento *1-n* não seria possível chegar em um resultado satisfatório, sendo necessário criar uma entidade associativa, vendas-produtos, que armazenaria as chaves primárias tanto de vendas como de produtos, portanto uma venda poderia ser associada a vários produtos e um produto ser associado a várias vendas. No final, com a criação da entidade associativa, o relacionamento voltará a ser *1-n,* como pode ser observado na **figura 1**. (Oppel, 2009).

Relacionamentos recursivos ocorrem quando uma entidade pode referenciar a uma instância dela mesmo, muito comumente usado em entidades de Categorias de produtos, onde dentro uma instância de categoria pode haver categorias filhas relacionada a uma categoria pai. (Oppel, 2009).

Figura 1 - Exemplo de Diagrama Entidade-Relacionamento



Fonte: Autoria própria

2.3 O MODELO RELACIONAL

Na década de 1970, Edgar Codd lançou um artigo denominado “Um modelo relacional de dados para grandes bancos de dados compartilhados”, trazendo então a ideia do modelo relacional para os bancos de dados. Muitas das transações de dados que são realizados todos os dias ocorrem porque Codd resolveu substituir o então modelo estrutural, ou hierárquico, por simples tabelas contendo linhas e colunas. [2]

Para Date (2004), a base da tecnologia no que refere a banco de dados, é o modelo relacional. O modelo relacional pode ser considerado o evento mais importante da área de administração de dados, muito porque desde a sua introdução, aproximadamente na década de 1970 quase todas as pesquisas sobre bases de dados são baseadas nesse modelo. O nome relacional é apenas um termo matemático para descrever uma tabela, visto que basicamente os dados que são exibidos para o usuário é uma tabela, ou então uma “nova” tabela extraída a partir de outras tabelas. A esse aspecto é dado o nome de estrutural, portanto em um

[2] IBM Archives: Edgar F. Codd. Edgar F. Codd. Disponível em: <http://www-03.ibm.com/ibm/history/exhibits/builders/builders_codd.html> Acesso em 12 de março de 2015.

modelo relacional tabelas é uma estrutura lógica, sendo a estrutura física armazenada livremente pelo sistema do modo que preferir, sequencialmente, indexando, utilizando *hashing*, etc.

2.4 SISTEMAS GERENCIADORES DE BANCOS DE DADOS

O SGBD é o responsável por ser a camada de software entre o usuário do sistema e os dados fisicamente armazenados. Ele controla as requisições de acesso ao banco, sejam elas para adicionar, remover, atualizar ou consultar os registros, atuando então de maneira muita parecida às linguagens de programação, sendo capaz de isolar o usuário do nível do *hardware*. O SGBD também suporta operações *SQL* (linguagem de consulta estruturada), desde que seja um sistema voltado para bancos relacionais, tornando-se o software mais importante do sistema. Por SGBD também costumam ser chamados alguns produtos genéricos, como o *Mysql, Oracle, PostgreSQL*, *Firebird* e etc., mas algumas pessoas tendem a chamar erroneamente esses sistemas apenas de bancos de dados, visto que banco de dados é o arquivo físico armazenado em disco, conforme explica Date (2004).

Oppel (2009) explica que um SGBD fornece serviços básicos necessários para a organização e manutenção de uma base de dados, por exemplo: mover os registros do modelo físico para o modelo lógico, quando necessário, gerenciar acesso concorrente aos dados por múltiplos usuários, incluindo prevenir possíveis conflitos de atualizações, prover mecanismos de segurança para impedir que dados sejam acessados e modificados sem autorização, permitir que cópias sejam feitas e restaurações quando necessárias, além de suportar a linguagem de consulta estruturada, conhecida como *SQL.*

2.5 A LINGUAGEM SQL

A *SQL* é a linguagem padrão ao se usar o modelo relacional, sendo desenvolvida pela equipe da IBM na década de 1970, para seus próprios produtos, sendo mais tarde adotada por muitos outros fornecedores. Hoje a linguagem *SQL* é completa contendo instruções como *while, leave, loop, call, return* e etc., mas a sua ideia original era de que fosse apenas uma sub linguagem de dados. (Date, 2004).

A *SQL* pode ser dividida em duas principais categorias:

DDL (*Data Definition Language*), que são declarações para criar, modificar e excluir objetos da base de dados, alguns exemplos:

CREATE, comando usado para criar objetos: *CREATE TABLE* nome_tabela; Comando *ALTER*, pode ser utilizado para trocar o tipo de um atributo em uma tabela e o comando *DROP*, que é utilizado para excluir objetos. (Oppel, 2009).

DML (*Data Manipulation Language*), onde ações conseguem modificar dados armazenados em tabelas, sendo seus principais comandos: *INSERT, UPDATE, DELETE* e *SELECT.* (Oppel, 2009).

Com o comando *INSERT* é possível inserir registros em uma tabela, com a sintaxe:

INSERT INTO <nome_tabela> *VALUES* (valor1, valor2,... valor n).

Já o operador capaz de realizar consultas na base de dados, é o *SELECT*. Com ele é possível realizar consultas na sua forma mais simples (*SELECT * FROM* <nome_da_tabela>), retornado então todos os registros das tabelas selecionadas. Como também é possível usar condições para filtrar os dados, como por exemplo:

*SELECT * FROM* Clientes *WHERE* data *BETWEEN* ‘1985-01-01’ *AND* ‘1989-01-01’, que retornaria todos os clientes nascidos entre as datas de 01/01/1985 e 01/01/1989. (Oppel, 2009).

O comando *UPDATE* serve para atualizar os dados já existentes, por exemplo: suponha que o *funcionário x* teve sua data de admissão gravada de maneira errada, então com o simples comando:

UPDATE funcionário *SET* data_admissao = ‘01/01/2000’ *WHERE* ID = 1 é possível atualizar esse campo, e graças à restrição *WHERE* não afetar os registros dos outros funcionários. (Oppel, 2009).

Com a instrução *DELETE*, é possivél excluir registros como:

DELETE FROM funcionários *WHERE* data_admissao < ‘01/01/2000’, que excluiria todos os registros de funcionários que foram admitidos antes da data proposta. (Oppel, 2009).

É importante ressaltar que os comandos realizados têm a ideia de ser durável, o que significa que após a exclusão de um dado, não é mais possível recuperá-lo, a menos que tenha sido feito uma cópia da base de dados recentemente. Em alguns casos indica-se também a não exclusão definitiva dos dados, adicionando um campo do tipo valor verdadeiro, onde o usuário ao excluir o

registro fosse realizado um *UPDATE* atualizando o valor do campo para falso e deixando de ser exibido para o usuário, mas ainda assim existindo no banco de dados, criando posteriormente uma rotina em uma linguagem de programação para excluir esses dados de maneira definitiva, quando ficasse comprovado que era desnecessário manter essa informação a salvo.

Sobre esse tema, Date (2004) relata que a importância da recuperação de dados ser designada ao usuário, como fazer uma cópia da base de dados de tempo em tempo, ocorre com mais frequência em bancos de pequeno porte, pois SGBDs mais robustos possuem esquemas de recuperação que são capazes de retornar o banco a um estado estável antes da aparição da falha, como um erro decorrido durante uma transição, por exemplo.

2.6 TRANSAÇÕES

Transação é uma unidade lógica de trabalho, que possibilita a execução de várias ações sob o mesmo comando, como transferir dinheiro de uma conta para a outra, que a principio pode parecer uma mesma ação, mas que na realidade são duas atualizações, uma para subtrair o saldo da onde saiu o dinheiro, e a outra para adicionar ao saldo da conta recebedora. O problema é que diversos fatores podem impedir que essa transação seja executada em sua totalidade, como uma queda no servidor onde o banco de dados está instalado. Porém graças ao gerenciador de transações é possível garantir que em caso de falha durante uma transação, as ações que foram executadas com sucesso sejam desfeitas, esse “fenômeno” recebe o nome de atomicidade. (Date, 2004).

A operação *COMMIT* avisa ao banco de dados que todas as ações de uma determinada transação foram bem sucedidas e que o banco de dados pode aceitar essas informações e salva-las. Já a operação *ROLLBACK* informará ao banco que alguma ação falhou e que todas as ações anteriores devem ser desfeitas. Essa recuperação é possível, pois o banco mantém uma espécie de diário, conhecido como *log*, onde ele registra todas as operações, especialmente o valor de um registro antes e depois de ser atualizado. Ao admitir que todas as ações foram executadas com sucesso o banco é capaz de persistir os dados, ou seja, torná-los permanentes. (Date, 2004).

No que concerne à consistência de dados, o sistema simplesmente assume que todas as transações não violam nenhuma restrição de integridade. (Date, 2004).

Ainda sobre transações, Date (2004) também afirma que elas devem garantir a atomicidade (realizar tudo ou nada), a correção (a transação quando executada encontra o banco de dados em estado correto e deve deixá-lo assim após sua finalização), o isolamento (uma transação só pode ver o resultado de outra transação após a segunda ser finalizada) e a durabilidade (após o sistema aceitar que todas as ações que foram executadas de maneira correta, ele deve ser capaz de manter as atualizações ainda que o mesmo venha a falhar em seguida). Esse termo também é conhecido genericamente, como *ACID*.

2.7 INTEGRIDADE

A integridade é uma restrição e pode ser considerada uma expressão booleana, que também pode ser uma regra de negócio do banco, como por exemplo, ao dizer que toda compra necessita de um fornecedor ou que uma determinada peça nunca pode ter quantidade em estoque menor que um valor pré-determinado. (Date, 2004)

Todas as restrições necessitam ser declaradas ao SGBD, que para garantir que essa restrição seja respeitada será necessário a ele ficar monitorando todas as operações executadas no banco e que contenham alguma restrição. No caso de uma restrição ser criada, mas no registro já houver alguma informação que infrinja essa restrição, ou seja, faça com que essa informação seja considerada como falsa, essa restrição será rejeitada ou poderá ser aceita e sua regra valerá a partir desse ponto (cada SGBD tem a maneira de lidar com essa situação). (Date, 2004).

O simples fato de designar um tipo a uma coluna, como um campo *ID* ser somente numérico ou um campo de data aceitar somente registros que sigam o padrão do banco de dados para data (*aaaa-mm-dd*) já é uma restrição, pois os valores que podem ser aceitos por essa variável precisam necessariamente ser do tipo que foi designado a ela.

2.8 DESVANTAGENS DO MODELO RELACIONAL

Bancos de dados relacionais podem se tornar custosos e problemáticos quando o volume de dados há ser armazenado e a quantidade de usuários que irá solicitar informações da sua base forem altas. Outro fato que contribui contra banco de dados relacionais é a impedância de dados. A incompatibilidade de impedância ocorre porque há diferença entre o modelo relacional e como funciona a estrutura de dados em uma memória. Como já mencionado em bancos de dados relacionais, os dados são estruturados em tabelas e linhas, que apesar de funcional e simples também contêm limites. A começar pelo valor de uma coluna que deve ser simples, não existindo a possibilidade de usar dados aninhados e comparado com a estrutura de dados da memória, isso é prejudicial, visto que a memória suporta estruturas mais ricas. Como ao observar um relatório de pedidos, que a princípio aparenta ser uma estrutura única, porém na realidade ocorre uma série de junções de tabelas para ser possível chegar aquele resultado, como por exemplo, a partir de registro do pedido, unir a tabela de itens dos pedidos, a tabela de itens, a tabela de clientes, a tabela de endereções de entrega, a tabela de forma de pagamento, e etc. (Sadalage *et al*, 2013).

Há de se ressaltar que após o surgimento de *frameworks* que realizam o mapeamento objeto-relacional, como o *Hibernate*, ficou mais fácil de lidar com a impedância em dados, mas o trabalho que esses *frameworks* poupam às pessoas pode comprometer o desempenho quando os desenvolvedores de aplicativo ignoram o banco de dados. (Sadalage *et al*, 2013).

Date (2004) lamenta que os atuais SGBDs que atuam com o modelo relacional, não são capazes de o fazerem em sua totalidade, adaptando-se durante os anos e deixando algumas de suas ideias de lado.

Neste capitulo pôde ser observado à importância de os dados serem armazenados e organizados de forma lógica e de como os SGBDs atuam na intermediação entre uma requisição solicitada por um aplicativo a sua real alteração na base de dados. Pôde ser observado também que o modelo relacional foi o marco mais importante no que se refere à modelagem e organização dos dados nos últimos

trinta anos, principalmente pela sua capacidade de garantir a integridade dos dados, mas também com a existência de obstáculos quando se trabalha com muitos dados e/ou a quantidade de requisições ao servidor for elevada, e que a partir do início da década de 2000 pela quantidade de dados gerados tornou-se necessário pensar em novas soluções para armazenamento de dados.

3 BANCOS DE DADOS NOSQL

Com o crescimento do comércio eletrônico a partir do inicio da década de 2000, o aumento no número de usuários com acesso a internet e o número de dispositivos com essas características, cresceram também a quantidade de dados que deveriam ser armazenados. As empresas que atuavam *online* começaram a registrar novos tipos de dados sobre seus usuários e a fazer um melhor mapeamento dos mesmos, como preferencias, produtos buscados, produtos visitados, histórico de compras e etc. Bancos de dados relacionais não foram desenvolvidos para trabalharem em *clusters* e a quantidade cada vez maior de dados faria com que as empresas aumentassem sua infraestrutura, também conhecido com escalabilidade vertical, ou seja, quanto maior a quantidade de dados, o poder de processamento e armazenamento das máquinas também teriam que crescer. (Sadalage *et al*, 2013).

A origem do *NOSQL* pode ser creditada a uma conferência que ocorreu no ano de 2009 nos Estados Unidos. Movidos pelos exemplos do *BigTable* (*google*) e *Dynamo* (*amazon*), muitos projetos foram iniciados com a ideia de armazenamento alternativo de dados. Com o objetivo de unir boa parte dessa pessoal, essa convenção foi proposta e eles decidiram que ela precisaria de um nome com as mesmas características da *hashtag* do *Twitter*, curto, que conseguisse passar a ideia e fácil de lembrar. Após algumas sugestões o termo *NOSQL* foi o escolhido, mas até hoje há discussões sobre seu significado, há quem defenda que *NOSQL* significaria *Not Only SQL* (não apenas *SQL*), porém é necessário entender que o termo *NOSQL* não precisa ser um acrônimo para algo, ele simplesmente é um termo, achado ao acaso que acabou se popularizando e descrevendo uma nova tendência de armazenamento de dados. (Sadalage *et al*, 2013).

Já para Tiwari (2011), o nome *NOSQL* provavelmente deveria passar a ideia de *NORDBMS,* ou seja, de que o modelo é não relacional. Algumas tentativas de fazer o nome *NOREL* ganhar adeptos foram feitas, mas sem sucesso, tornando a opção *Not Only SQL,* a mais comum e difundida por pessoas que atuam com esse modelo.

Tiwari (2011) também diz que NOSQL não é o símbolo de um único produto, mas representa uma classe de produtos e uma coleção de conceitos sobre armazenamento e manipulação de dados.

As principais características de bancos de dados *NOSQL* são: **Escalabilidade horizontal**: um número maior de máquinas é disponibilizado para realizar o armazenamento e processamento dos dados. **Esquemas flexíveis:** não há, ou há de maneira bem pequena, esquemas pré-definidos sobre sua estrutura de dados e essa ausência de esquema auxilia a escalabilidade de dados, mas também diminui a integridade dos mesmos. **Replicações:** existem dois tipos principais de replicações: *Master-Slave* (**Mestre-Escravo**), onde a escrita do banco é feita no nó mestre e refeita a partir dele em todos os nós escravos. Em caso de falha do nó mestre, um novo nó assume essa posição e o processo pode continuar. Esse tipo de replicação é principalmente recomendado quando houver requisições de leitura e não recomendado quando o trafego de inserções e atualizações forem altos. Lembrando que os usuários podem ver dados inconsistentes, pois ao acessar determinada aplicação que está relacionada a um nó que eventualmente falhou e ainda não recebeu os dados atualizados do mestre, a consistência de dados estará prejudicada. O outro tipo de replicação é o **ponto a ponto**, onde todos os nós tem o mesmo valor, tornando possível que todos recebam os dados de gravações e caso algum falhe, não impedirá o armazenamento de dados, mas o problema de dados inconsistentes permanece, pois quando é possível armazenar em mais de um local, usuários podem acessar o mesmo registro e tentar atualizá-lo ao mesmo tempo, ocorrendo então um conflito. Caso esse conflito seja de gravação, e não de leitura, o problema de inconsistência torna-se mais perigoso. (Sadalage *et al*, 2013).

Para uma gravação ser considera consistente, não é necessário que todos os nós aceitem a atualização, mas sim que a maioria deles, ou em alguns casos torna-se necessário mesclar as informações. Em todas as hipóteses, a consistência está sendo “relaxada” para um ganho de desempenho. (Sadalage *et al,* 2013).

Há ainda a opção de se usar bancos de dados *NOSQL* em um único servidor, mesmo que muitos bancos de dados *NOSQL* sejam projetados para ser executados em muitos servidores, sempre que for possível, para evitar problemas de dados inconsistentes, além da menor dificuldade e complexidade do projeto por parte dos desenvolvedores, habituados a planejarem seus bancos dessa maneira, é recomendada a utilização em um único servidor. (Sadalage *et al*, 2013).

Outra característica de bancos de dados *NOSQL*, é a **consistência eventual,** sendo necessário muitas vezes sacrificar a consistência, principalmente quando se está trabalhando com muitos nós. O teorema *CAP* (*Consistency*, *Availability and Partition tolerance*), introduz a ideia de que dado três itens: *consistência*, *disponibilidade* e *tolerância a partições*, é possível garantir apenas dois deles. Um banco de dados armazenado em um único servidor pode ser consistente e disponível, mas não será possível utilizá-lo com partições. Já em um sistema particionado, será necessário balancear a consistência e a disponibilidade, para atender as necessidades especificas de cada projeto. (Sadalage *et al*, 2013).

Além do que já foi dito, bancos de dados *NOSQL* não utilizam a linguagem *SQL*, apesar de alguns modelos usarem uma linguagem de consulta muito semelhante ao *SQL*, e também por, na maioria das vezes serem código aberto. (Sadalage *et al*, 2013).

É importante também ressaltar que, por atuarem sem esquemas, banco de dados *NOSQL* são muito mais flexíveis, permitindo serem adicionados campos aos registros, sem a necessidade de alterar a estrutura. Isso evita campos definidos no modelo relacional, mas que são necessários apenas em algumas regiões. Por exemplo: uma empresa de atuação global de desenvolvimento de jogos precisa armazenar o código do *CPF* dos jogadores brasileiros, porém esse dado só faz sentido se a pessoa for brasileira e essa empresa mantém armazenados os registros de 10.000 (dez mil) pessoas cadastradas a sua base de dados e apenas 1.000 (um mil) são brasileiras, restando então 9.000 (nove mil) registros onde o campo *CPF* ficaria vazio, enquanto no modelo *NOSQL* é possível armazenar esse campo apenas se for realmente necessário. Apesar de ser um exemplo superficial serve para ilustrar a capacidade de flexibilidade dos modelos de bancos de dados *NOSQL*.

Sadalage, et al (2013) ressaltam que o surgimento do modelo *NOSQL*, não implica que o modelo relacional está com os seus dias contados, visto que ele continuará sendo muito eficaz dependendo do tipo de aplicação e do público alvo que o desenvolvedor deseja atingir, além da familiaridade que a maioria dos desenvolvedores tem com ele. A partir do crescimento dos bancos de dados *NOSQL*, há uma nova opção para armazenamento de dados, conhecido também como persistência poliglota, ou a capacidade de utilizar diferentes formas de armazenar os dados dependendo de cada situação, como por exemplo, como os dados serão manipulados.

A falta de padrão é uma desvantagem do modelo de banco de dados *NOSQL*, visto que por não ter esquema definido e trabalhar muito próximo ao aplicativo é necessário um cuidado maior ao alimentar agregados, pois um campo criado em um agregado com nome *qtd* para referenciar quantidade será diferente de um campo com o nome de *quant*, e se isso não for tratado no sistema, será prejudicial.

Marcadores de versões auxiliam o SGBD quando se está trabalhando com múltiplos nós, pois ao solicitar uma informação, pode ocorrer de duas respostas diferentes serem recebidas, e os marcadores de versões auxiliaram ao SGBD sobre qual é o mais recente. Marcadores de versões podem ser um contador, um *timestamp* (hora atual do servidor que realização a operação), etc. (Sadalage *et al*, 2013).

Sistemas *NOSQL* cresceram rapidamente e hoje em dia são respaldados por empresas como, *Google*, *Facebook*, *Amazon* e *Linkedln*, e foram criados para suprir limitações do modelo relacional. Sendo as principais razões para preferir por um modelo *NOSQL*: grande quantidade de usuários, grande quantidade de dados, computação em nuvem e “a internet das coisas” (*the internet of things*).[3]

3.1 MAPEAR-REDUZIR

Dados executados em *cluster* tem a vantagem de poder dividir o processamento dos registros entre muitas máquinas, mas esses dados também podem ser volumosos e diminuir a quantidade que será transferida pela rede torna-se muito importante. (Sadalage *et al,* 2013).

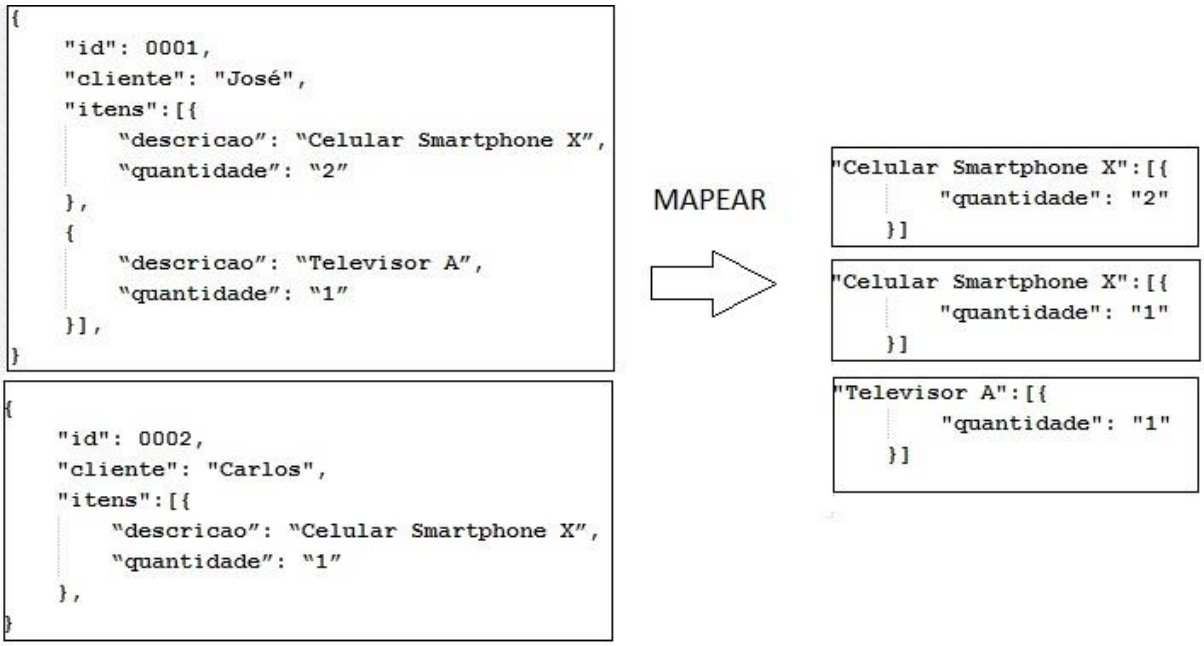
Segundo Dean *et al* (2004), usuários especificam um mapa que após processado transforma-se em pares de chave/valor que têm os seus valores mesclados baseado em seu atributo chave com o objetivo de reduzir o retorno dos dados.

Sadalage *et al* (2013) cita como exemplo uma lista de agregados de Pedidos, onde os itens de uma compra são armazenados aninhados ao pedido, sendo isso necessário porque os clientes assim veem os seus pedidos. Porém em determinada situação a equipe de vendas deseja conhecer a quantidade de determinados itens que foram vendidos, mas isso poderia ser custoso visto que seria necessário

[3] Couchbase, Why NoSQL? Disponível em: <http://www.couchbase.com/nosql-resources/what-is-no-sql> Acesso em 15 de maio de 2015.

acessar todos os agregados, entrar em todas as listas e só então chegar ao resultado desejado. Contudo como observado na **figura 2**, à função *map* lê os registros dos bancos de dados, e os transformam em pares de chave valor.

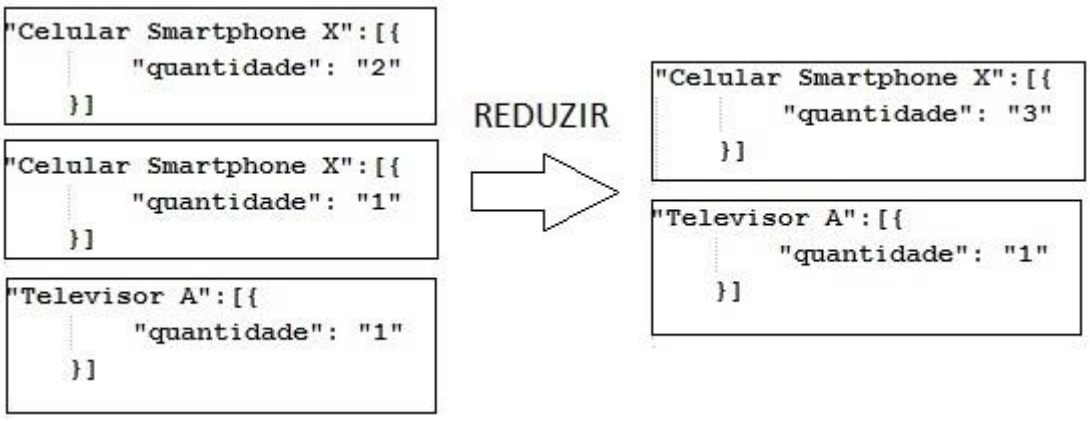
Figura 2 - Exemplo de mapeamento de uma pesquisa



Fonte: Autoria própria

Na figura 3, a função *reduce* agrega todos os pares de chave valor com a mesma chave.

Figura 3 - Exemplo de redução de uma pesquisa



Fonte: Autoria própria

3.2 MODELO AGREGADO DE DADOS

O modelo de agregados de dados possibilita que se trabalhe com dados mais complexos dos que os oferecidos pelo modelo relacional, como por exemplo, armazenar um conjunto de listas em vez de referenciar outra tabela que armazenaria essas informações. Das quatro principais categorias de bancos de dados *NOSQL*, três utilizam o modelo agregado de dados: chave-valor, documento e famílias de colunas. Como agregado pode se entender como um conjunto de objetos relacionados que precisam ser tratados com uma única unidade. Bancos de dados que atuam com modelo de dados agregados também são importantes para trabalhar em *cluster*, pois esse conjunto de objetos facilita a replicação e a fragmentação. (Sadalage *et al*, 2013).

Banco de dados orientados a agregados suportam transações atômicas dentro de um agregado, por isso a importância de como será a estrutura do agregado, que podem variar de projeto para projeto. (Sadalage *et al*, 2013).

O formato *JSON* (*notação de objetos javascript*) é o mais comum aceito quando se faz necessário representar informações do modelo *NOSQL*, como na **figura 4**, onde pôde ser observado o agregado Clientes que contém os dados principais, mas também permite armazenar uma lista de endereços.

Figura 4 - Formato de escrita *JSON* para representar dados *NOSQL*

```
{
 "id": 1,
 "nome": "José",
"endereço":[
        {
        "rua": "Rua Exemplo 1",
        "numero": "1",
        "cidade": "Azul"
        },
        {
        "rua": "Rua Exemplo 2",
        "numero": "20",
        "cidade": "Amarelo"
        }
    ],
}
```

Fonte: Autoria própria

A forma como o agregado será organizado dependerá de cada caso, é possível criar dois agregados, para Clientes e Pedidos, e dentro do agregado de

Pedidos armazenar uma lista de Itens, como também pode ser criado um único agregado de Clientes, que contém uma lista de pedidos, e dentro de cada pedido uma lista de itens. Se o objetivo do agregado for listar todos os pedidos de um determinado cliente, então a melhor opção é a segunda, porém quando houver a necessidade de listar a quantidade total de um determinado item a aplicação terá de percorrer todos os agregados, abrindo lista por lista para poder chegar ao número desejado.

3.3 BANCO DE DADOS CHAVE-VALOR

Esse modelo trabalha principalmente com agregados, onde cada um desses agregados contém uma chave ou *ID* que serve para diferenciá-lo. Ele é dividido em duas colunas, a primeira que serve para armazenar a sua chave e a segunda o valor, não importando de qual tipo seja esse valor, sem alguma restrição esse campo aceita valores como *blob*, texto, *JSON*, *XML*, e etc. No SGBD Riak (que trabalha com o modelo de chave-valor) chaves podem ser segmentadas em *Buckets,* que é um termo criado por esse SGBD para diferenciar as chaves. (Sadalage *et al*, 2013).

Quando um banco chave-valor está sendo executado em diversos nós, sua consistência depende de vários fatores, na maioria dos casos os SGBDs optam por deixar como padrão a opção de que uma operação é considerada válida e consistente quando ela consegue ser replicada para mais da metade dos nós existentes. Exemplo: casa haja cinco nós, se os dados atualizados conseguirem se replicar por três deles, então ela é considerada consistente. Existe também a possibilidade na criação de um *Bucket* de informar ao banco que gravações nesse *Bucket* só são consideradas estáveis se for aceita por todos os nós. Caso haja conflito de informações por eventual falha de sistema de algum nó que impossibilitou a gravação, o banco exibe as duas informações para o usuário e permite que ele tome a decisão sobre qual registro deve ser mantido. A mesma ideia se aplica as transações. A quantidade de nós que são necessários para validar uma transação, pode ser inserida manualmente ou usar a padrão do SGBD. (Sadalage *et al*, 2013).

As consultas em bancos de chave-valor ocorrem apenas pela chave de cada *Bucket,* pois como a coluna valor aceita qualquer tipo de dado, estruturado, ou não, uma busca dentro do campo torna-se bastante complexa, e a maior parte dos

SGBDs que atuam com esse modelo não a implementam. Até por esse motivo, esse modelo de banco de dados não é recomendado quando for necessário relacionar os dados ou realizar buscas mais complexas pelos dados inseridos na coluna valor. (Sadalage *et al*, 2013).

Esse modelo pode ser escalado usando a fragmentação, utilizando o nome da chave para diferenciar em qual nó ele deve pertencer. Exemplo: Caso existam cinco nós e que todas as chaves iniciem com letras de *A* a *E*, então quando um determinado registro começando com a letra A for ser armazenado ele será encaminhando para o seu respectivo nó. Caso esse nó venha a falhar, todos os registros que contenham a inicial que seria designada para ele, não conseguirão completar a tarefa.

3.4 BANCO DE DADOS DE DOCUMENTOS

Bancos de dados de documentos são capazes de armazenar e recuperar estruturas de dados como *XML*, *JSON*, sendo a sua organização baseada na forma de árvores hierárquica. Os documentos armazenados tendem a ter semelhanças entre si, mas não necessariamente serão iguais. Como no exemplo de **figura 5**, onde os documentos contêm características diferentes, mas que pertencem à mesma coleção, não existindo também dados nulos, se um valor não for adicionado para determinada característica, presumisse que essa característica é dispensável. O que não ocorreria em um a SGBD relacional, onde todas as linhas de uma mesma tabela contêm o mesmo esquema e valores não transmitidos para determinado campo é registrado como nulo. (Sadalage *et al*, 2013).

Figura 5 - Estrutura de banco de dados de documentos

```
{
    "id": 1,
    "cliente": "José",
    "endereco": "Rua dos ipes, 200",
    "telefone": "(19) 1111 - 1111",
}

{
    "id": 2,
    "cliente": "Maria",
    "endereco": [{
        "rua": "dos laranjais",
        "complemento": "Fundos",
        "bairro": "Floresta"
    }]
}
```

Fonte: Autoria própria

Cada instância de um sistema gerenciador que utiliza o modelo de documentos pode conter vários bancos de dados, cada banco de dados pode contar múltiplas coleções, e cada coleção armazenar estruturas semelhantes, mas não idênticas de dados. Como na **figura 5**, onde os dois documentos fazem parte da coleção de Clientes.

Para Harrison (2010), bancos de dados orientados a documentos foram criados para atuar em uma área onde o modelo relacional não consegue, que a diferença entre como bancos de dados relacionais são organizados, e como isso é feito em aplicativos que utilizam a linguagem de programação orientada a objetos, apesar de existirem *frameworks* que automatizam esse mapeamento, o esforço necessário para realizar esse mapeamento apenas serve para amenizar a dor. Enquanto que em uma base de dados orientada a documentos, os documentos podem ser desenvolvidos de maneira praticamente semelhante à linguagem de programações. Porém em documentos, dados são muitas vezes duplicados, visto que, por exemplo, em uma coleção de Pedidos, na lista de itens seria necessário armazenar muitas vezes os nomes dos produtos, o que geraria problemas de integridade, caso um produto tivesse algum dado alterado.

3.5 BANCOS DE DADOS FAMÍLIA DE COLUNAS

Nesse modelo de armazenamento de dados, bancos de dados são conhecidos como *keyspaces,* tabelas são famílias de colunas e linhas e colunas mantém a mesma nomenclatura, porém no caso de colunas elas podem diferenciar de uma para outra, dado a necessidade. Famílias de coluna são conjuntos de dados capazes de serem relacionados e que podem ser acessados juntos. (Sadalage *et al,* 2013).

Esse modelo foi introduzido primeiramente pelo *Google,* e até por isso muitas vezes mencionado como o modelo de dados *BigTable*. Sendo as principais características o particionamento de dados, consistência elevada, porém não é capaz de garantir alta disponibilidade. (Teorema CAP, apenas dois dos três itens são possíveis de garantir). (Pokorny, 2011).

Os registros são indexados por um conjunto, conhecido também como tripla, a linha, coluna e *timestamp*, onde o *timestamp* é usado para diferenciar versões de um mesmo registro. (Sadalage *et al*, 2013).

Muitas vezes esse modelo é comparado ao modelo relacional, mas enquanto no modelo relacional, os dados são armazenados em linhas, no modelo de famílias de colunas, os dados são armazenados em colunas. Outra diferença considerável entre os modelos é os fatos de que colunas podem ser adicionadas em determinadas linhas, sem que todas as linhas de uma mesma família necessitem receber também essas colunas. (Sadalage *et al*, 2013).

Alguns SGDBs que atuam com famílias de colunas, utilizam linguagem de consulta muito parecida a *SQL*. Como o Cassandra, que possui uma linguagem como o nome de *CQL* (*Cassandra Query Language*), sendo possível criar famílias de colunas, *CREATE COLLUMNFAMILY* <família_coluna>, pode-se também selecionar todos os registros, ou apenas as colunas que precisar com um comando *SELECT*, que também pode receber condições para filtrar dados, como *WHERE,* o seja, comandos similares ao de bancos de dados que trabalham no modelo relacional. (Sadalage *et al*, 2013).

A escalabilidade também pode ser conseguida facilmente adicionando mais nós, pois nenhum nó é considerado mestre e ao adicionar mais nós, apenas melhorará o desempenho e a capacidade de gravações e leituras. (Sadalage *et al*, 2013).

3.6 BANCO DE DADOS DE GRAFOS

Nesse modelo é possível armazenar os nós, que possuem propriedades, e os seus relacionamentos, que são representados por arestas que podem conter propriedades (o nó no modelo de grafos é diferente dos nós descritos nos modelos anteriores, que serve para descrever quando as informações estão sendo compartilhadas em diversos servidores). Alguns relacionamentos tem sentido bidirecional, como por exemplo, o relacionamento denominado “amigo” que liga a propriedade nome, com o conteúdo *Jonatas* (nó) ao outro nó com propriedade de nome *José*, implica tanto que *Jonatas* é amigo de *José*, como que *José* é amigo de *Jonatas*, porém nem todos os relacionamentos são assim, o relacionamento que indica que *Jonatas* curtiu uma *foto* não *implica* que a foto curtiu *Jonatas*. Sendo preciso informar o banco quando uma relação é bidirecional. (Sadalage *et al*, 2013).

Uma consulta em grafo é conhecida como travessia, sendo uma das vantagens de se utilizar esse modelo à possibilidade de alterar o valor de uma propriedade sem alterar os nós ou as arestas. Já nos relacionamentos também é possível guardar informações, como por exemplo, em um relacionamento que indica uma amizade é possível armazenar quando essa amizade foi iniciada. (Sadalage *et al*, 2013).

Modelos de dados baseados em grafos não suportam a distribuição em servidores, sendo possível usá-lo apenas em um único servidor, a consistência é alta, (alta consistência, alta disponibilidade, não distribuído) sendo importante também ressaltar que nós só podem ser excluídos se não houver relacionamentos entre eles. (Sadalage *et al*, 2013).

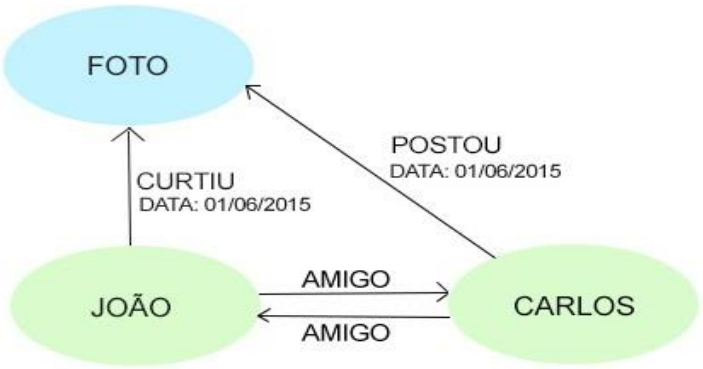
Esse modelo de dados suporta linguagens de consultas, e tanto os seus nós, como seus relacionamentos podem ser indexados, tornando possível a pesquisa ser relacionada pelos itens indexados. Pode se também filtrar informações baseadas nas direções dos seus relacionamentos. Como por exemplo, é possível saber todo mundo que curtiu uma determinada foto, ou todas as fotos que uma determinada pessoa curtiu. Bancos de dados de grafos têm a capacidade de determinar qual o menor caminho para chegar a determinado nó. (Sadalage *et al*, 2013).

Os seus usos mais comuns são onde os dados precisam estar conectados, como em redes sociais. Mecanismos de recomendação de produtos, como “quem viu este produto, viu também...”, ou “seus amigos recentemente curtiram...”. Além de

serem apropriados para informar quais produtos geralmente são comprados junto com o produto que um cliente está comprando no momento. (Sadalage *et al*, 2013).

Na **figura 6** pode ser observado o exemplo de um relacionamento usando grafos para representar uma parte de uma rede social, onde um usuário *curtiu* uma foto que foi *publicada* por outro.

Figura 6 - Exemplo de grafos



Fonte: Autoria própria

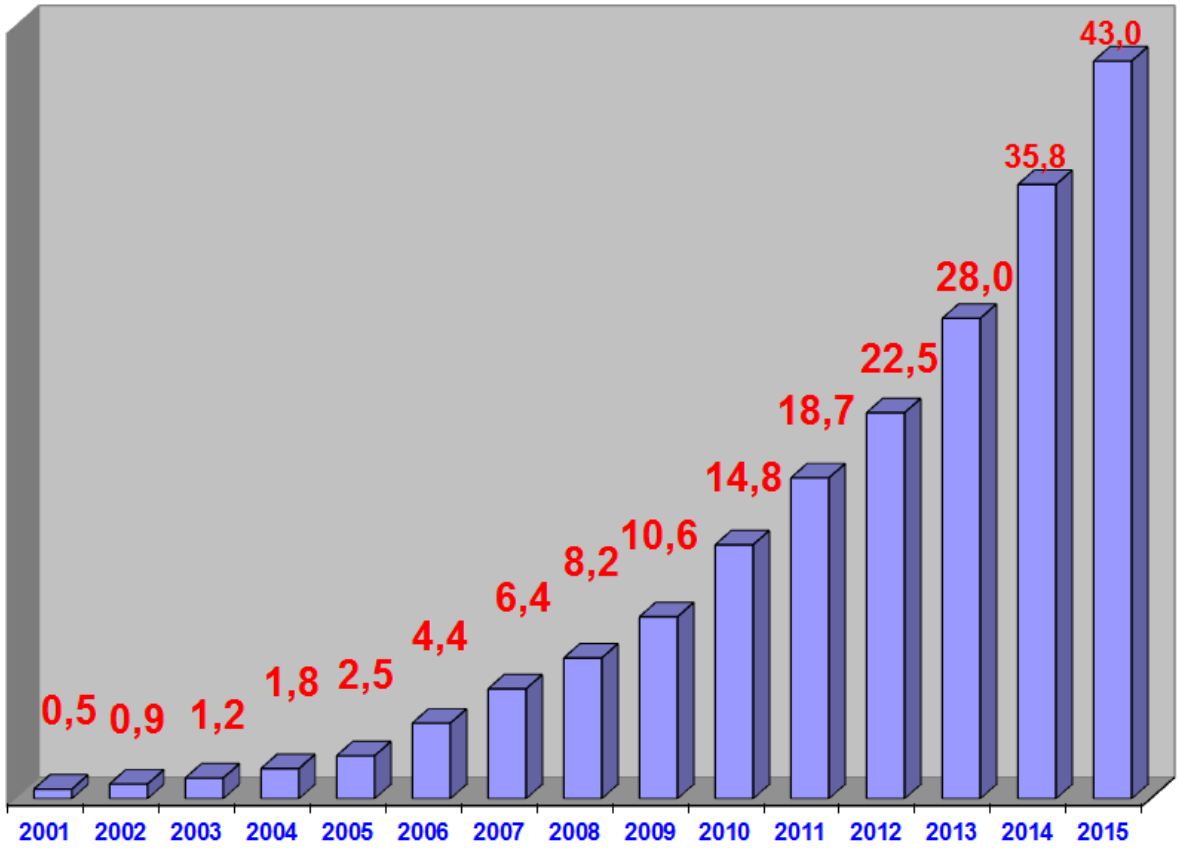
Esse capítulo visou conceituar as principais características do modelo *NOSQL*, bem como os principais modelos de bancos de dados que se enquadram e podem ser intitulados como *NOSQL*. Pôde ser observado também, que um modelo *NOSQL*, apesar do que o nome pode indicar, em alguns casos poderá dispor de uma linguagem de consulta bem parecida com o *SQL* e que o seu principal objetivo é suprir as necessidades de um mundo cada vez mais globalizado e onde o número de dados requisitados e transferidos pela internet é crescente. No próximo capítulo será possível observar a modelagem de um banco de dados para um site de vendas pela internet utilizando os conceitos de uma base de dados não relacional.

4 ESTUDO DE CASO DE UM COMÉRCIO ELETRÔNICO

O ramo de negócio escolhido para desenvolver um modelo de banco de dados *NOSQL* foi o de comércio eletrônico, pois como pode ser observado na **figura 7**, seu faturamento pode chegar à casa dos 50 bilhões de reais no ano de 2015, porém alguns transtornos ainda acontecem com quem tenta acessar ou realizar uma

compra em um site, principalmente em datas especiais, como próximo ao natal, ou no *Black Friday*. Alguns pedidos não são finalizados, clientes não conseguem acessar a determinado site e são empurrados para uma fila, aguardando a conexão ser liberada para ter acesso às compras, entre outros transtornos.

Figura 7 - Faturamento anual de e-commerce no Brasil



Fonte: http://www.e-commerce.org.br/

O modelo *NOSQL* escolhido para armazenar os principais dados do sistema foi o de documentos, pois suas características, de dados aninhados e a facilidade de se moldar agregados vão ao encontro com a estrutura de dados que um site de venda eletrônico armazena. O modelo de grafos foi escolhido para armazenar os dados de uma pequena parte do sistema, a que indicará produtos para outros usuários com base em visitas e compras, por ser um sistema de banco de dados cujos atributos facilitam esse trabalho.

4.1 ESCOPO DO ESTUDO DE CASO

Antes de tudo é preciso definir as principais características que esse site de comércio eletrônico irá conter. Lembrando que os principais sites de vendas contêm uma estrutura que aparenta ser mais complexa do que a apresentada no estudo de caso, pois precisam armazenar mais informações e também validar mais dados, além do que cada aplicativo contém sua regra de negócio que poderá gerar dados específicos. Sendo o objetivo desse estudo de caso passar a ideia central de como funcionaria um banco de dados *NOSQL* voltado para um comércio eletrônico.

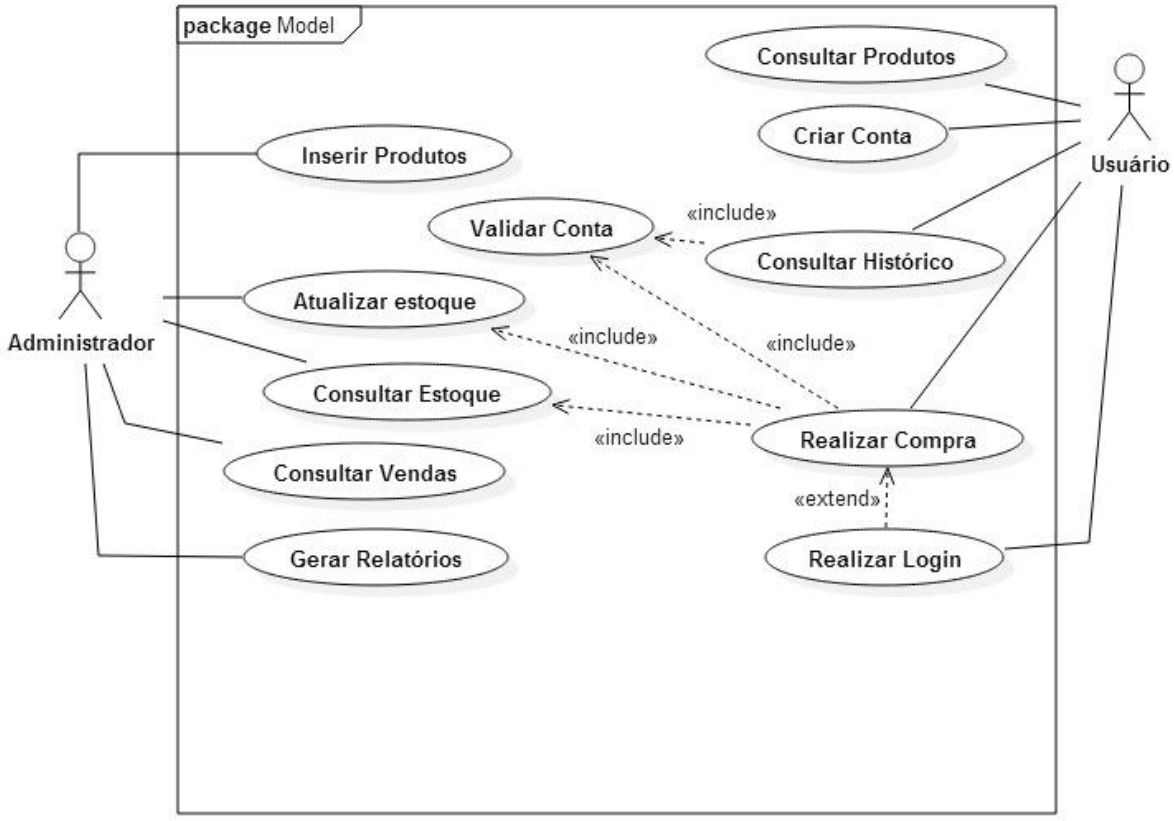
O objetivo desse site em questão é vender produtos pela internet, e para isso é preciso armazenar algumas informações como dados dos produtos, fornecedores, clientes, categoria de produtos, e também os dados das transações de vendas de produtos, e o histórico de compras do cliente e da empresa (pois a empresa compra os produtos dos fornecedores para ser vendidos no site).

É preciso salientar a existência da figura do administrador do site (que pode ser uma pessoa, ou uma equipe), que alimentará os dados do sistema, inserindo produtos, atualizando estoque, cadastrando fornecedores, gerando relatórios gerenciais para apoio a tomada de decisão e etc. Há ainda a existência de outro ator em um cenário de comércio eletrônico que é o cliente, que pode acessar os produtos sem necessariamente ter uma conta, ou ter realizado o *login*, como também pode criar o cadastro e efetuar compras, além de consultar seu histórico.

Essas ações podem ser melhores observadas na **figura 8**, em que é utilizado um diagrama de caso de uso para exemplificar as principais funções do sistema. O caso de uso é um diagrama UML (*Unified Modeling Language*) que auxilia na modelagem e documentação de um sistema.

A figura do administrador e do usuário é representada por atores que estão ligadas a toda ação que são capazes de realizar no aplicativo. (ações são armazenadas dentro das elipses ligadas aos atores).

Figura 8 - Caso de uso do sistema



Fonte: Autoria própria

O “comando” *extend* indica que alguma outra ação pode ser executada quando a primeira, já o “comando” *include* indica que havendo aquela ação, necessariamente haverá a ação relacionada, como por exemplo, quando o usuário for realizar uma compra, obrigatoriamente será necessário validar a conta do usuário.

4.2 MODELANDO AS COLEÇÕES

Definido os objetivos do sistema é possível modelar as coleções que armazenarão os dados do aplicativo. Apesar do conceito do *NOSQL* visar bancos de dados sem esquema, a modelagem de dados torna-se importante, pois minimizará erros, visto que apontará os principais campos que cada agregado deverá conter e também as suas devidas nomenclaturas, pois com um descuido do desenvolvedor, ou então caso dois aplicativos estejam acessando a mesma base de dados, seria possível incluir um campo “*nome*” e um “*nomeCli*” ambos para referenciar-se ao

nome do cliente, e o banco de dados aceitaria ambos, gerando possíveis erros no futuro. Além de toda modelagem, tanto do aplicativo como do banco de dados, são fatores importantes no processo de criação de um sistema.

Figura 9 - Coleção de fornecedores

```
{
    _id: <ObjectId>
    nome: "Fornecedor exemplo 1",
    cnpj: "11.111.111/0001-11",
    ie: "ISENTO",
    endereco: [
        { rua: "Exemplo 1 - 100", bairro: "Bairro Exemplo 1", cidade: "Americana", estado: "SP"}
    ],
    contato: [
        { nomeContato: "José da Silva", telefone: "(11) 1111-1111", email: "forncedor1@fornecedores.com.br"}
    ]
}
```

Fonte: Autoria própria

Na **figura 9** é possível observar a primeira coleção, a de fornecedores, nesse exemplo é possível ver os principais dados sendo armazenados diretamente no agregado e os dados “compostos” como endereços e contatos armazenados como listas, tornando possível armazenar inúmeros deles.

Figura 10 - Coleção de clientes

```
{
    _id: <ObjectId>,
    nome: "Armando Silva",
    cpf: "111.111.111-11",
    email: "armando.silva@clientes.com.br",
    senha: "12346",
    historico:[
        {
            pedidos: [
                {
                    notaFiscal: "124242",
                    dataEmissao: "01/02/2014",
                    valorTotal: 2.350,00,
                    status: [
                        {dataStatus: "15/02/2014", descStatus: "Entregue"}
                    ],
                    itens: [
                        {descricao: "Aparelho Celular A", valor: 1.000,00, quantidade: 1, totalItem: 1.000,00},
                        {descricao: "Televisor A", valor: 1.350,00, quantidade: 1, totalItem: 1.350,00}
                    ]
                }
            ]
        }
    ]
}
```

Fonte: Autoria própria

O agregado de clientes é representado na **figura 10**, com a mesma ideia do de fornecedores, podendo ser adicionado dados aninhados se necessário. O diferencial nesse exemplo é que o histórico de compras do cliente é também

armazenado em listas dentro do seu perfil, pois ao acessar seus dados, todo seu histórico é carregado, evitando que possíveis consultas futuras sejam necessárias.

Figura 11 - Coleção de categorias

```
{
    _id: <ObjectId>,
    categoria: "Eletrônicos",
    descricao: "Os melhores produtos eletrônicos."
}
```

Fonte: Autoria própria

A **figura 11** representa a coleção de categorias, que possui uma estrutura mais simples.

Figura 12 - Coleção de itens

```
{
    _id: <ObjectId>,
    fornecedor_id: <ObjectId>,
    categoria_id: <ObjetciId>,
    nome: "Aparelho celular",
    descricao: "descrição do Aparelho (...)",
    cor: "Prata",
    altura: "altura em mm",
    largura: "largura em mm",
    comprimento: "comprimento em mm",
    caracteristicas: "Dados complementares",
    valores: [
        {
            valorCompra: 500,00,
            valorVenda: 1.000,00
        }
    ],
    estoque: [
        {
            quantidadeMinima: 10,
            quantidadeAtual: 25
        }
    ],
    impostos: [
        (dados de impostos especificos para cada tipo de produto podem ser declarados aqui)
    ]
}
```

Fonte: Autoria própria

A **figura 12** representa o modelo de como seria a coleção de Itens. Como os itens não conterão sempre as mesmas características alguns campos não aparecerão em determinados registros. Já outros dados, como valor de compra e valor de venda, tornam-se imprescindíveis para que a empresa saiba informações como o saldo que foi arrecadado.

Figura 13 - Coleção de Pedidos

```
{
    _id: <ObjectId>,
    cliente_id: <ObjectId>,
    dataEmissao: "01/04/2014",
    dataSaida: "01/04/2014",
    horaSaida: "13:40:00",
    itens: [
        {item_id: <ObjectId>, valorItem: 1.000,00 , quantidadeItem: 1, valorTotal: 1.000,00},
        {item_id: <ObjectId>, valorItem: 32,00 , quantidadeItem: 2, valorTotal: 64,00},
    ],
    valorTotal: 1.064,00,
    tipoEntrega: "PAC",
    valorFrete: 7,00,
    valorTotalFrete: 1.071,00,
    observacoes: "Dados adicionais da nota fiscal"
}
```

Fonte: Autoria própria

A última coleção, representada na **figura 13**, é a de Pedidos, que novamente contém dados aninhados para representar informações que são repetidas, como os itens de uma compra. Nesse exemplo, o valor de cada item, é repetido, apesar de já ser armazenado no agregado de itens, vale a pena armazená-lo novamente, pois apesar de um comércio eletrônico ser impessoal, já é possível em alguns sites o usuário negociar o valor do produto, além de em outros casos o item ter seu valor alterado, como em promoções, e em casos de alta procura de um determinado item o preço do mesmo pode subir.

Um fator importante a salientar é que em alguns casos é criado um campo para referenciar um item de outro agregado, assim como no modelo relacional, como no agregado de pedidos, onde os itens, e o cliente são referenciados pelo seu identificador. Portanto, ao desenhar um agregado, é preciso ter em mente, qual o objetivo e o quanto a capacidade da infraestrutura estarão disponíveis. Pois ao se tomar uma decisão de duplicar os dados, como no caso da coleção de clientes, os dados do histórico dos clientes são informações repetidas, visto que já são armazenadas no agregado de Pedidos. Não sendo o problema a capacidade de armazenamento, essa seria uma opção para evitar múltiplas consultas e ganhar desempenho.

4.3 BANCO DE DADOS MONGODB

O MongoDB é um banco de dados de código aberto, orientado a documentos desenvolvido para fácil desenvolvimento e escalabilidade. E exatamente pelo seu modelo ser orientado a documentos e um dos principais desse gênero, que foi o escolhido pelo estudo de caso, sendo selecionado para serem apresentadas algumas de suas funções e características.[4]

No sistema gerenciador de bancos de dados mongoDB, agregados são conhecidos como coleções, e a cada coleção criada é gerado automaticamente um identificador de 12 *bytes*. Esse identificador é único, porque contém informações do segundo em que foi criado, como um identificador da máquina, um identificador do processador e um contador de números escolhidos randomicamente. O mongoDB dispõe de drivers que possibilitam sua conexão com várias linguagens de programação, como *Java*, *C++*, *Phyton* e *PHP*, e todos os seus comandos podem também ser executados em um editor conhecido como *mongo Shell Edition*. Esse sistema gerenciador é possível escalar o banco de dados horizontalmente, o que significa que caso seja necessário mais espaço em disco para armazenar os dados, é só adicionar mais máquinas, também conhecidos como nós.[5]

4.3.1 PRINCIPAIS OPERAÇÕES

Como na maioria dos SGBDs relacional, com o mongoDB é possível realizar as principais operações de um banco de dados, como inserção, atualização, pesquisa e remoção de dados.

A criação de uma coleção não precisa ser declarada explicitamente, ao inserir as informações e indicar uma coleção que ainda não existe, ela é automaticamente gerada. Porém, caso o desenvolvedor ache necessário, existe a possibilidade de declarar a coleção. Já os campos que cada coleção irá conter não são declarados, em cada inserção em uma coleção, os campos ali dispostos são admitidos. Por isso

---

[4] MONGODB MANUAL. Introduction to MongoDB. Disponível em: <http://docs.mongodb.org/manual/core/introduction/> Acesso em 20 de abril de 2015.

[5] MONGO DB. MongoDB Overview. Disponível em: <http://www.tutorialspoint.com/mongodb/mongodb_overview.htm> Acesso em 22 de abril de 2015.

a necessidade de ao utilizar um banco de dados não relacional, é necessário criar certas regras para evitar erros, como pode ser visto na **figura 14**. [6]

Figura 14 - Exemplo de criação de coleção e inserção

```
1. db.createCollection(<nome da coleção>)

2. db.<nome da coleção>.insert("nome": "nome exemplo1")

3. db.<nome da coleção>.insert("nomeUsuario": "nome exemplo1")
```

Fonte: Autoria própria

Na **figura 15** é mostrado um exemplo prático, utilizando a coleção de cliente para a inserção os dados.

Figura 15 - Exemplo de inserção na coleção de clientes

```
db.clientes.insert( "nome":"Armando Silva", "cpf":"111.111.111-11",
                    "email":"armando.silva@clientes.com.br",
                    "senha":"123456",
                    "pedidos":[
                        {
                            "notaFiscal":"124242", "dataEmissao":"01/02/2014", "valorTotal": 2350.00,
                            "status": [
                                {"dataStatus":"15/02/2014", "desStatus":"Entregue"}
                            ],
                            "itens":[
                                {"descricao":"Aparelho Celular A", "valor": 1000.00, "quantidade": 1, "totalItem": 1000.00},
                                {"descricao":"Televisor A", "valor": 1350.00, "quantidade": 1, "totalItem": 1350.00}
                            ]
                        }
                    ])
```

Fonte: Autoria própria

Utilizando o mongoDB como SGBD, para realizar a operação de atualização de um registro é necessário utilizar uma sintaxe como essa:

db.<nome da coleção>.*update*(<condição>, <dados a serem atualizados>). Por exemplo, para atualizar um registro da coleção de clientes:

db.clientes.*update*({“_id”:”<ObjectId>”, “nome”: ”Armando da Silva Moraes”}).

Vários campos de um registro podem ser alterados de uma vez, mas caso a clausula passada na condição não exista, então é inserido um novo registro.

A sintaxe para remover dados é:

db.<nome da coleção>.remove(condição), como por exemplo: db.clientes.*remove*({“email”:”armando.silva@clientes.com.br”}).

---

[6] MONGO DB. MongoDB Insert Document. Disponível em: <http://www.tutorialspoint.com/mongodb/mongodb_insert_document.htm> Acesso em 22 de abril de 2015.

Se nenhuma informação for passada como condição, então todos os dados da coleção são removidos.

Para realizar consultas a sintaxe mais simples é a que retorna todos os registros da coleção: db.<nome da coleção>.*find*().

Há também comandos para auxiliar a filtragem de dados, já que com algumas exceções, a maioria das consultas em um sistema tem condições para filtrar os dados. Como por exemplo:

db.clientes.*find*.*limit*(10), que limitará a quantidade de retorno de dados. É possível realizar comandos como soma, média, menor valor, maior valor, o primeiro e o último.[7]

4.3.2 MAPEAR E REDUZIR UMA COLEÇÃO NO MONGODB

Como mencionado no capitulo três, trabalhar com agregados tem a desvantagem na hora de realizar consultas, principalmente para relatórios, pois para exibir a quantidade total vendida de um determinado item pode significar ter de entrar em todos os pedidos, passar por toda a lista de itens um a um para saber o resultado. Porém com a técnica também já mencionada de mapear e reduzir torna-se possível facilitar esse trabalho, e o banco de dados mongoDB também permite que seja possível realizá-lo em seu SGBD.

A **figura 16** representa a estrutura de um mapeamento e redução para obter a quantidade de registros vendidos e com status marcado como entregue. Supondo que todos os campos existam na coleção em questão, então será retornado o identificador e a quantidade total dos itens.

Figura 16 - Mapeando e reduzindo pedidos

```
db.pedidos.mapReduce(
                        function() { emit( this.item_id, this.quantidadeItem); },
                        function(key, values) { return Array.sum(values)},
                        {
                            query: {descStatus: "Entregue"},
                            out: "Totais"
                        }
                    )
```

Fonte: Autoria própria

[7] MONGODB. MongoDB Query Document. Disponível em: <http://www.tutorialspoint.com/mongodb/mongodb_query_document.htm> Acesso em 22 de abril de 2015.

4.3.3 REPLICAÇÃO DOS DADOS NO ESTUDO DE CASO

Replicações são importantes no modelo não relacional, pois com os dados mantidos em máquinas diferentes é possível proteger a base de dados de perdas de falhas de nós específicos, além de auxiliar a capacidade de leitura. No mongoDB replicações são do tipo mestre-escravo, ou seja, um servidor primário recebe os dados e os replica para os outros nós. No caso de falha do servidor primário, então um dos nós é eleito como primário e todo o processo pode continuar. Apenas o servidor primário é capaz de receber solicitações de escrita, já a função dos servidores secundários, além de replicar os dados também é possível solicitar requisições de leitura. É possível também solicitar que alguns servidores secundários sejam designados para nunca serem primários, e também para nunca receberam as replicações, apenas *backup* de dados.[8]

4.4 UTILIZAÇÃO DA PERSISTÊNCIA POLIGLOTA

Cada necessidade de um aplicativo pode ser suprida por diferentes bancos de dados. A ideia da persistência poliglota é utilizar para cada dificuldade diferente de um aplicativo uma modelagem de dados que se adeque exatamente a essa situação, e não fazer com que um mesmo banco de dados se adeque a todas as características do sistema.

No exemplo do estudo de caso de um comércio eletrônico, isso poderia ser aplicável na parte em que é possível sugerir ao usuário que leu a ficha técnica de determinados produtos, outros produtos que usuários que consultaram o mesmo produto que ele, também visitaram. E esse modelo de dados, é o de grafos, pois através das suas arestas que interligam duas ações é possível extrair essa informação, e outras, como: “Quem comprou esse produto, também comprou...”.

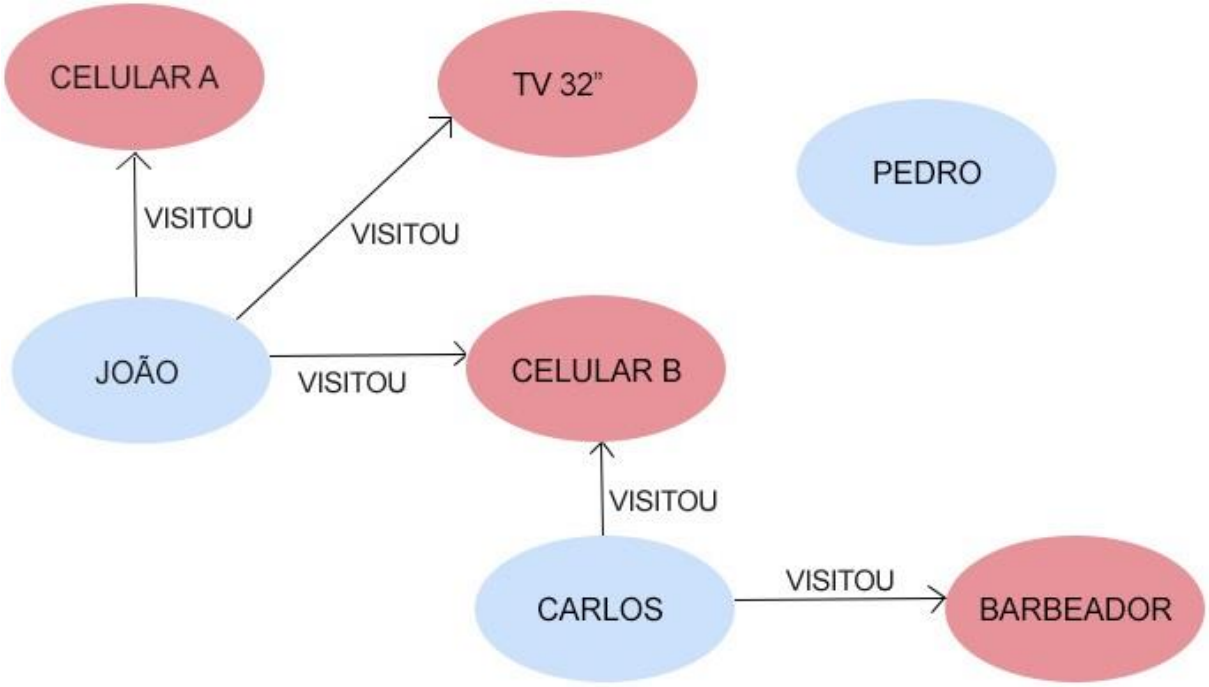
É preciso ter em mente, que o modelo de dados em grafos, não foi desenhado para trabalhar em cluster de computadores, diferentemente do modelo de documentos. Por isso é necessário pensar em estratégias de armazenamento. O fato do SGDB escolhido para o estudo de caso (mongoDB), trabalhar com a situação de mestre escravo, onde só o servidor principal aceita informações de escrita auxilia

[8] MONGODB MANUAL. Replication Introduction. Disponível em: <http://docs.mongodb.org/manual/core/replication-introduction/> Acesso em 20 de abril de 2015.

esse trabalho, pois evita que informações sejam perdidas, e alguns dados poderiam deixar de ser armazenados no banco de dados de grafos.

Na **figura 17** é possível observar uma situação onde um usuário, Pedro, ainda não visitou nenhum produto, enquanto outros usuários já visitaram.

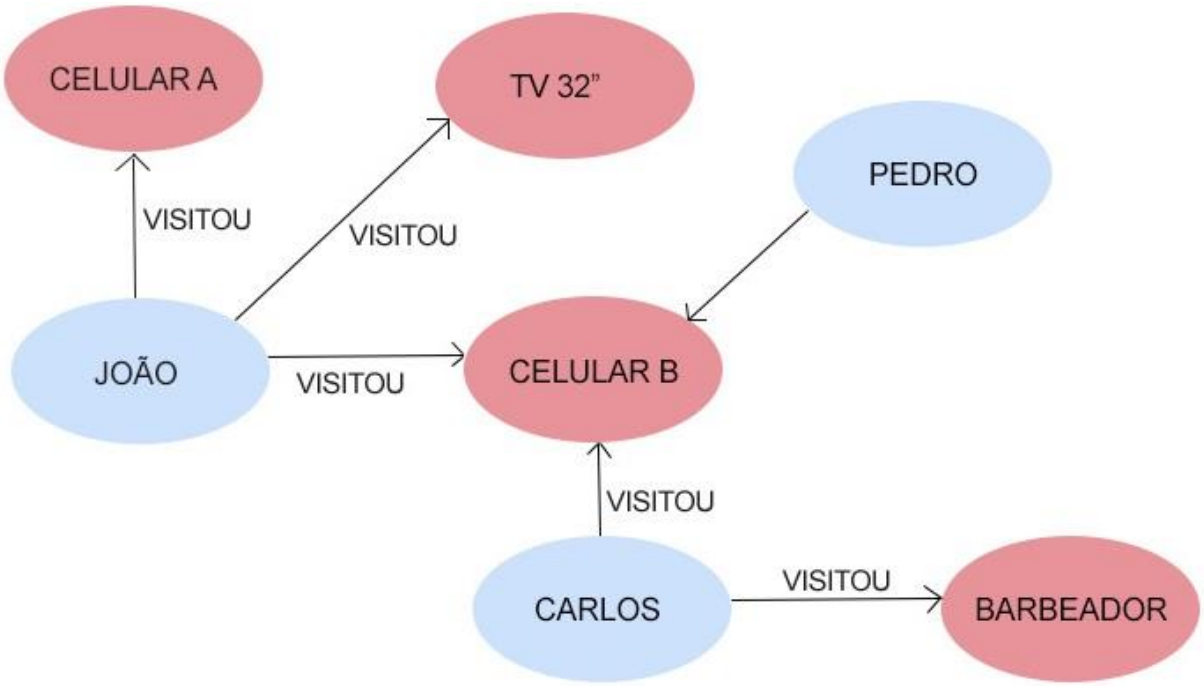
Figura 17 - Banco de dados de grafos antes da visita



Fonte: Autoria própria

Na **figura 18**, o usuário *Pedro* visita um produto (*Celular B*), e a partir do relacionamento que é criado, é possível indicar a eles outros produtos que usuários que também visitaram o mesmo produto que ele, visitaram.

Figura 18 - Banco de dados de grafos depois da visita



Fonte: Autoria própria

4.4.1 O SGBD NEO4J

O gerenciador de banco de dados NEO4J, é um dos mais conhecidos e usados para se trabalhar no modelo de grafos e possuem duas versões, uma para conhecimento da ferramenta, gratuita, e outra para utilização em empresa que após um período de teste, passa a ser paga. Contém também diversos fóruns de apoio que auxiliarão no desenvolvimento de projetos. Com um banco de dados de grafos, por exemplo, é possível indicar a um usuário que acessa o site novos produtos para visitar, baseando-se em produtos visitados por outros usuários que visitaram algum item em comum com ele.[9]

No gerenciador NEO4J, a sintaxe para a criação de nós e os relacionamentos poderia ser descrita assim:

[9] NEO4J. Graph Data Modeling Guidelines. Disponível em: <http://neo4j.com/developer/guide-data-modeling/> Acesso em 18 de maio de 2015.

CREATE (<nome do nó>: <"tipo" do nó> {<nome da propriedade>: <valor da propriedade>}).

Podendo existir quantas propriedades forem necessárias. Por exemplo:

CREATE (jonatas: Usuario {nome: 'Jonatas'}),

CREATE (celular_A: Produtos {nome: 'Celular Exemplo A'}).

Para dizer ao banco de dados que o usuário Jonatas visitou o produto '*Celular Exemplo A*' o comando seria:

CREATE (jonatas) –[:VISITADO {data_visita: 01/04/2014}] -> (celular_A).

Em sistemas de gerenciamento de grafos também é possível percorrer os nós e verificar as informações que se deseja obter, como por exemplo: Saber quais produtos diferentes usuários também visitou a partir de um produto. Então para saber quais produtos, usuários que visitaram o produto '*celular_A*' também visitaram deve-se usar o comando:

MATCH (produtos :Produtos {nome: 'Celular Exemplo A'}) - ->(<-[: VISITADO])-(nome) *return* nome.

É possível ainda criar diversas consultas, como saber quais foram os produtos mais visitados, ou então, em quais casos houve a chamada compra "casada", isto é, quando um produto é vendido com um item complementar, como uma capa de banco para carros e a capa para volante.

4.5 O ESTUDO DE CASO NO MODELO RELACIONAL

No modelo relacional, esse mesmo estudo de caso seria organizado de maneira diferente, tornando necessário criar mais tabelas, como as entidades fracas e as associativas, por exemplo, para ser possível realizar a normalização. Algumas tabelas fracas também seriam necessárias, como: a de telefone para clientes e fornecedores, a de endereço também para ambos, e etc., além de entidades associativas, como pedidos produtos.

4.6 CONCLUSÕES SOBRE O ESTUDO DE CASO

O presente estudo de caso demonstrou que utilizar um banco de dados não relacional pode ser uma boa opção pela sua flexibilidade e variedade de opções que permitirão ao desenvolvedor conseguir chegar ao seu resultado final ao começar a

criação de um *software*, adaptando seu banco de dados a sua necessidade, e não ao contrário como ocorre na maioria das vezes. A persistência poliglota também torna-se uma possibilidade ao abrir a mente para novas ideias, fazendo sua base de dados cada vez mais adaptável as necessidades. Entretanto utilizar um banco de dados relacional continua sendo uma opção razoável ao desenvolver um sistema e em conjunto com bancos não relacionais o ganho pode ser ainda melhor.

Antes da escolha da utilização de sistemas gerenciadores não relacionais em um projeto, é preciso saber das dificuldades e dos problemas ao optar por replicar os dados em nós e saber que em alguns casos usuários poderão ver dados desatualizados, por isso vale o planejamento para definir quais informações deverão ter alta consistência e quando a consistência poderá ser relaxada em relação ao ganho do desempenho.

5 CONSIDERAÇÕES FINAIS

Essa pesquisa buscou proporcionar uma nova visão sobre gerenciamento de banco de dados e uma forma de ampliar o conhecimento de pessoas ligadas ao desenvolvimento de sistemas que estão ou não diretamente relacionadas à escolha do gerenciador de banco de dados, sua modelagem e desenvolvimento, pois diferentemente dos gerenciadores relacionais, quem optar por utilizar um sistema não relacional para criar sua base de dados utilizará uma linguagem que estará fortemente interligada à linguagem de desenvolvimento tradicional do sistema.

Bancos de dados NOSQL podem ser adicionados à rotina de desenvolvimento por equipes que já tenham vivência na área de criação e que já tenham trabalhado em projetos anteriores, porém como toda nova tecnologia, bancos de dados não relacionais têm uma curva de aprendizado que necessita ser respeitada. Para poder utilizá-la será necessário antes um trabalho de pesquisa e levantamento de informações sobre o objeto especifico. Será também necessário avaliar se é realmente necessário utilizar essa ferramenta colocando todos os prós e contras e chegar à conclusão se o resultado obtido utilizando sistemas *NOSQL* será melhor que utilizando bancos relacionais. É recomendado antes de colocar um projeto contendo um banco de dados não relacional em produção que haja um período de testes para melhor avaliação.

Bancos de dados que utilizam conceitos *NOSQL* têm a vantagem de serem flexíveis, de fácil adaptação ao sistema, escalabilidade horizontal, possibilidade de replicação em vários servidores, além de quando a quantidade de dados a ser armazenada for grande. Porém esse conceito também tem as suas desvantagens em relação ao modelo relacional, entre elas: dificuldade de migrar dados existentes em bancos relacionais, falta de padronização, dados inconstantes no servidor devido a queda de algum nó, sem contar a possível dificuldade que equipes que adaptadas a trabalharem com o modelo relacional teriam ao se deparar com as ideias e inovações do modelo *NOSQL*.

Como toda tecnologia, bancos de dados não relacionais precisam passar por um período de maturidade e de certa forma alguma padronização, apesar de bancos

não relacionais, como já visto, deixar o desenvolvedor bem flexível em relação à moldagem de seu banco de dados.

O objetivo geral foi alcançado, pois proporcionou várias opções e ideias para modelar bancos de dados de maneira diferente do que propõe o modelo relacional. A hipótese correta é a de que sistemas *NOSQL* podem ajudar a resolver problemas com bancos de dados, porém a mão de obra especializada pode ser mais difícil de encontrar e possivelmente mais cara. O estudo se justificou porque possibilitou a ampliação do conhecimento na área de banco de dados.

Para trabalhos futuros sugere-se que bancos de dados não relacionais sejam avaliados na pratica e com os resultados dos testes obtidos chegarem à conclusão mais concreta sobre sua utilização em relação a bancos relacionais. Vale ressaltar que talvez para uma melhor analise seja necessário utilizar um banco de dados não relacional utilizando vários nós.

REFERÊNCIAS BIBLIOGRÁFICAS

_____. **Referências**: NBR-6023/ago. 2002. Rio de Janeiro: ABNT, 2002.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS - ABNT. **NBR 10520**: informação e documentação: citações em documentos: apresentação. Rio de Janeiro: ABNT, 2002. 7p.

Couchbase, **Why NoSQL?** Disponível em: <http://www.couchbase.com/nosql-resources/what-is-no-sql> Acesso em 15 de maio de 2015.

DATE, C. J. **Introdução a sistemas de bancos de dados**. 8º edição. Education Person, 2004.

DEAN, J; GHEMAWAT, S. **Map Reduce:** Simplified data processing large clusters. Disponível em: <http://static.googleusercontent.com/media/research.google.com/pt-BR//archive/mapreduce-osdi04.pdf> Acesso em 24 de abril de 2015.

GIL, A. C. **Como elaborar projetos de pesquisa.** 4° edição. São Paulo: Atlas, 2002.

HARRISON, G. **NoSQL and Document-Oriented Databases.** Disponível em: < http://www.dbta.com/Columns/Notes-on-NoSQL/NoSQL-and-Document-Oriented-Databases-72035.aspx> Acesso em 10 de abril de 2015.

IBM Archives: Edgar F. Codd. Edgar F. Codd. Disponível em: <http://www-03.ibm.com/ibm/history/exhibits/builders/builders_codd.html> Acesso em 12 de março de 2015.

Internet Users, **Internet Users in the World.** Disponível em: <http://www.internetlivestats.com/internet-users/> Acesso em 10 de maio de 2015.

MongoDB, **MongoDB Manual 3.0 Manual.** Disponível em: < http://docs.mongodb.org/manual/> Acesso em 20 de abril de 2015.

Neo4j, **Graph Data Modeling Guidelines.** Disponível em: < http://neo4j.com/developer/guide-data-modeling/> Acesso em 18 de maio de 2015.

OPPEL, A. J. **Databases: A beginner's guide.** The McGraw-Hill Companies Inc, 2009.

POKORNY, J. **NoSQL databases:** a step to database scalability in web environment. In: EMERALD GROUP PUBLISHING LIMITED. International journal of web. Information systems, 2013. p. 69-82.

SADALAGE, P. J; FOWLER, M. **NoSQL Essencial:** Um Guia Conciso para o Mundo Emergente da Persistência Poliglota. 1º edição. São Paulo: Novatec, 2013.

SETZER, V. W. ; SILVA, F. S. C. **Banco de dados: aprenda o que são**, **melhore seus conhecimentos**, **construa os seus**. Edgar Blucher, 2005.

TIWARI, S. **Professional NoSQL.** 1º edição. John Wiley & Sons Inc, 2011.

Tutoriais point, **MongoDB Tutorial.** Disponível em: <http://www.tutorialspoint.com/mongodb/> Acesso em 22 de abril de 2015.