CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Kayo Vidal Brighenti

O BANCO DE DADOS NOSQL COMO SOLUÇÃO PARA PROBLEMAS DE ESCALABILIDADE

Americana, SP

2014

CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Kayo Vidal Brighenti

O BANCO DE DADOS NOSQL COMO SOLUÇÃO PARA PROBLEMAS DE ESCALABILIDADE

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. M. Sc. José Alberto Florentino Rodrigues Filho.

Área de concentração: Banco de Dados

Americana, S. P.

2014

Brighenti, Kayo Vidal

B864b O banco de dados NoSQL como solução para problemas de escalabilidade. / Kayo Vidal Brighenti. – Americana: 2014.

47f.

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 Alberto Florentino Rodrigues Filho

1. Banco de dados I. Rodrigues Filho, José Alberto Florentino II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.3.07

Kayo Vidal Brighenti

O BANCO DE DADOS NOSQL COMO SOLUÇÃO PARA PROBLEMAS DE ESCALABILIDADE

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, 02 de dezembro de 2014.

Banca Examinadora:

José Alberto Florentino Rodrigues Filho
Mestre
Faculdade de Tecnologia de Americana

Clerivaldo José Roccia
Mestre
Faculdade de Tecnologia de Americana

Francesco Artur Petrotti
Mestre
Faculdade de Tecnologia de Americana

AGRADECIMENTOS

Em primeiro lugar agradeço ao meu orientador, professor M. Sc. José Alberto Florentino Rodrigues Filho, o qual foi essencial no desenvolvimento de todo conteúdo deste trabalho, pelo tempo reservado à sanar minhas dúvidas e pela disposição de me orientar sempre que foi necessário. Também agradeço a FATEC Americana por disponibilizar espaço e conteúdo intelectual para que eu pudesse enriquecer ainda mais este trabalho.

DEDICATÓRIA

Dedico primeiramente este trabalho aos meus pais e familiares, que sempre me deram força e suporte para eu continuar lutando pelos meus objetivos. Também dedico aos meus amigos que tiveram paciência e também deram suporte emocial para que eu pudesse estar inteiramente focado neste trabalho.

RESUMO

A escalabilidade de sistemas tem sido amplamente discutida no meio empresarial e ainda perpetuará por anos. Essa necessidade ocasionou no surgimento de um novo modelo de banco de dados, o NoSQL. Este, agora, não utiliza mais a linguagem dos modelos relacionais, e possuia linguagem própria para utilização de seus SGBD's. O NoSQL trouxe a possibilidade de se dividir o processamento de requisições, gravações e modificações de dados, sem prejudicar a disponibilidade do sistema, e é um dos modelos que coloca em risco a utilização de modelos relacionais pelas empresas. Ainda trouxe a possibilidade de se utilizar ambos modelos, tanto relacional, quanto não-relacional, em partes diferentes do sistema, tirando máximo proveito de cada modelo, alavancando desempenho e escalabilidade da aplicação desenvolvida. Escalando verticalmente, podemos adicionar novos *hardwares* à infraestrutura existente, e horizontalmente, adicionando novos servidores que forneçam maior disponibilidade ao sistema. O objetivo deste trabalho é provar que o NoSQL pode ser uma alternativa para sistemas que não escalam bem. O resultado observado é evidente, e o NoSQL cumpre seu papel de forma muito eficiente, oferecendo ferramentas para que os administradores de bancos de dados possam utilizar toda a capacidade dos modelos NoSQL.

Palavras Chave: escalabilidade; bancos de dados; não-relacional;

ABSTRACT

The scalability of systems have been widely discussed in the business environment and will perpetuate for years. This need resulted in the rise of a new model of databases, the NoSQL. Now, this didn't use more the relational model language, and own a language for the utilization of their SGDB's. The NoSQL brought the possibility to divide the processing of queries, recordings, and alterations of data, without sacrificing the availability of the system, and is the model that puts in risk the utilization of the relational model by the companies. Also brought the possibility of using the two models, both relational and non-relational, in different parts of the systems, taking full advantage of the two models, leveraging the developed application performance and scalability. Scaling vertically, we can add new hardware to the existing infrastructure, and horizontally, adding new servers that provides more availability to the system. The objective of this work is to prove that the NoSQL can be an alternative to the systems that don't scale well. The observed result is clear, and the NoSQL fulfills its role very efficiently, offering tools to databases administrators, which can use the full capacity of NoSQL models.

Keywords: *scalability, databases, non-relational*

SUMÁRIO

LISTA DE FIGURAS E DE TABELAS

1 INTRODUÇÃO

Vivemos em uma sociedade totalmente conectada a novas formas de tecnologia e acostumadas à rápida propagação das informações. A tecnologia vinculada à banco de dados vive hoje numa atualização dos modelos de estruturação. Da criação de novas formas de armazenamentos até a formulação de novos métodos de categorização de informações foram anos aplicados em pesquisas, trabalhos e testes. A questão de escalabilidade no âmbito tecnológico obrigou aplicações a migrarem da forma com que são executados. Essas, agora, não mais rodavam em um ambiente local, mas sim, à partir de um local remoto por meio de chaves, a nossa famosa *cloud*, ou nuvem. Embasada nessa situação, o banco de dados não-relacional surgiu como uma alternativa aos bancos de dados relacionais em prol da estrutura de armazenamento de dados que ora fora localmente armazenados e passavam a ser alocados horizontalmente em servidores espalhados pelo globo.

O termo escalabilidade tem sido constantemente discutido na área de banco de dados. Tem como significado a capacidade de usar sua total capacidade de armazenamento sem comprometer a qualidade e desempenho do sistema. Em tempos que empresas estão em constante expansão, nem sempre conseguem manter a escalabilidade de seus servidores em um nível aceitável, muitas vezes abdicando da escalabilidade para poder investir em outras áreas que possam trazer melhores resultados. Em contrapartida, o *downtime* e os riscos de se perderem dados são maiores. Em sistemas pequenos, a escalabilidade pode ser um fator crítico de desempenho. A falta de investimento em produtos e serviços que possam fornecer uma estrutura melhor, faz acentuar-se a questão da responsividade do sistema.

A escalabilidade está estritamente relacionada ao *hardware* empregado na sua construção. Ela aumenta diretamente de acordo com que o *hardware* instalado é atualizado ou acrescentado.

Dentro do termo, há vários métodos em que podem se medir o nível de escalabilidade que disponibiliza o hardware, dentre eles: a carga de escalabilidade, que consiste na viabilidade da expansão de sistemas conforme sua demanda cresce; a escalabilidade geográfica, que aborda a manutenção da utilidade e do desempenho, mesmo que o banco tenha sua extensão distribuída em outro ponto geográfico extremamente oposto ao original e; a escalabilidade administrativa, que, por sua vez

define a permanência da facilidade de uso e gerenciamento do banco, mesmo que este seja compartilhado entre várias instituições dentro de um sistema distribuído.

O *hardware* pode ser escalado de duas formas distintas: escalado verticalmente, de forma que se adicione mais ferramentas a um único serviço já existente do sistema, como a compra de um processador mais atualizado e potente, visando o seu desempenho; ou escalado horizontalmente, em que podem ser virtualizados servidores, adicionados computadores à estrutura existente, visando retirar a carga de um lugar e distribuí-la uniformemente com a adição de novos componentes.

O banco de dados relacional consegue fornecer um nível aceitável de escalabilidade quando bem aplicado. Contudo, para que se tenha um sistema altamente escalável, os investimentos devem ser sólidos. Garantir um sistema altamente escalável é garantir o menor *downtime* possível do sistema, um alto desempenho e melhor disponibilidade, fornecendo assim, um melhor serviço aos usuários de todo o sistema.

O modelo relacional tem como principal característica a confecção de tabelas para a constituição de relacionamento entre elas. Esse modelo mostrou-se flexível e capaz de resolver problemas que outros modelos de banco de dados não resolviam, adotando-se assim pela eficiência e facilidade do padrão. Essas relações que se formam entre tabelas são constituídas por atributos, ou campos, e mostram o tipo de dado que esse atributo irá representar. Ainda que não há um caminho previamente estipulado para acessar todas essas informações armazenadas. Esse padrão ainda oferece a atomicidade de dados, ou seja, caso haja algum erro nos dados fornecidos para serem inseridos ao banco, nenhum deles será efetivamente acrescentado.

Por outro lado, o modelo não-relacional destrói todo esse conceito de tabelas trazendo a opção de trabalhar horizontalmente com os dados, ou seja, não serão criadas tabelas para mais campos, todos os dados serão alocados no mesmo registro em sua totalidade. Na falta de espaço para novos registros, o NoSQL oferece como solução a adição de mais servidores, que não necessariamente precisam ser de grande porte e performance, para armazenar a quantidade necessária de dados. Além de serem tolerantes à falhas e oferecer suporte à clusterização de servidores. Tem como principal aplicação os sistemas em nuvem, e servidores que necessitam de um desempenho superior em consultas e escritas no banco.

O NoSQL é subdividido pelo tratamento dado ao seu conteúdo. Cada um deles oferece soluções apropriadas ao tipo de sistema que o utilizará. São suas subdivisões: Famílias de Colunas, Armazenamento de Documentos, Depósitos de Chave-Valor, Bancos de dados de Grafos, Modelos Agregados, que serão explicados individualmente no decorrer deste trabalho. Cada subdivisão supracitada, é gerenciada por um ou vários programas disponíveis para a gestão de dados.

O objetivo deste trabalho é provar que o banco de dados não-relacional é uma excelente ferramenta quando os níveis de escalabilidade e desempenho estão longe de atingirem o mínimo necessário ao sistema. Quando se farão necessários, quando serão alternativas aos modelos relacionais, quando não trarão tantos benefícios, por quê deveria ser utilizado no sistema. A importância da mudança de modelos relacionais para os novos conceitos e os gerenciadores disponíveis para o NoSQL também serão tratados minuciosamente.

A metodologia usada para a composição deste trabalho monográfico teve como base pesquisas virtuais e bibliográficas.

2 BANCOS DE DADOS

Primeiramente, é preciso que se explique o que são dados antes de analisarmos a fundo os bancos de dados.

2.1 DADOS

Dados são a representação de fatos, conceitos e informações coletadas à partir de fontes externas e que serão necessárias à tarefa a ser realizada.

Habitualmente, dados são confundidos com informações. Informação nada mais é do que dados organizados e sequenciados de forma legível e útil. Dados são percebidos pelo nosso corpo, processados, e a junção de um ou vários dados gera uma informação.

A informação no cenário atual é de vital importância para empresas e organizações. Armazená-las significa, de forma geral, conhecer o seu cliente e adaptar-se aos seus interesses. A quantidade e qualidade das informações armazenadas podem ser o diferencial para se alcançar o sucesso.

Alguns fatores são levados em conta quanto à eficiência e qualidade dos dados.

Atualidade do dado: atualmente, o tempo de validade de um dado tem continuamente encurtado, e torna-se um dos fatores mais importantes. É importante para o sistema, sempre armazenar um dado atualizado, garantindo assim, que as decisões a serem tomadas sejam eficientes e eficazes.

Correção: não basta só a atualidade do dado, se ele não for verdadeiro.

Relevância: o excesso de informações também pode ser uma dificuldade. Para que isso não ocorra, é necessário verificar a importância daquele dado para seu armazenamento. Armazenar dado sem relevância é prejudicial, e pode levar à decisões errôneas.

Disponibilidade: ainda que fique certificado que os três fatores anteriores estejam de acordo, a disponibilização do dado deve ser imediata. Decisões tomadas sem dados disponíveis geram um risco enorme à organização.

Legibilidade: o dado só poderá ser interpretado e processado, quando puder ser lido. Nenhum dos outros fatores poderão ser efetivados se uma informação ou um dado não puderem ser lidos.

Ainda assim, dados podem ser apresentados de três formas distintas. Dados quantitativos são aqueles que apresentam quantidades e são utilizados nas funções aritméticas do sistema. Dados qualitativos são aqueles que descrevem ou especificam, e complementam os dados quantitativos. Dados referenciais são aqueles que permitem gerenciar e referenciar os elementos abordados.

2.2 BANCOS DE DADOS

Basicamente, bancos de dados são sistemas capazes de armazenar dados, oriundos de análises e informações.

Date (2003, p.3-6) define o que é um sistema de banco de dados da seguinte maneira:

> Um sistema de banco de dados é basicamente apenas um sistema computadorizado de registros. O banco de dados, por si só, pode ser considerado como o equivalente eletrônico de um armário de arquivamento; ou seja, ele é um repositório ou recipiente para uma coleção de arquivos de dados computadorizados.
> Os sistemas de banco de dados estão disponíveis em máquinas que variam desde pequenos computadores de mão (hand-held) ou computadores pessoais até os maiores mainframes ou clusters de computadores de grande porte. Nem é preciso dizer que os recursos fornecidos por qualquer sistema são determinados, até certo ponto, pelo tamanho e pela potência da máquina sendo utilizada.
> Os bancos de dados são integrados e (normalmente) compartilhados; eles são usados para armazenar dados persistentes. Esses dados podem ser – de modo útil, embora informal – considerados representações de entidades, juntamente com relacionamentos entre essas entidades – apesar do fato de que um relacionamento seja, na realidade, um tipo essencial de entidade.

Ainda, em sua obra, faz uma breve análise sobre dados e os modelos existentes de bancos de dados:

> Existe outra (e importante) maneira de pensar sobre o que são realmente os dados e os banco de dados. A palavra dado vem da palavra latina datu, que corresponde a “dar”; portanto, os dados são na realidade fatos dados, a partir dos quais podemos deduzir fatos adicionais.
> O modelo relacional não é o único modelo de dados; existem outros – embora a maioria deles seja diferente do modelo relacional pelo fato de serem ad hoc, até certo ponto, em vez de terem uma base sólida, assim como o modelo relacional é baseado na lógica formal. Nesse caso, surge a pergunta: em geral, o que é um modelo de dados? (DATE, 2003, p. 13 - 14)

Ad hoc nesse contexto significa que são interligados por relacionamentos, principalmente no modelo relacional.

E embasado nessas informações, Date (2003, p. 31-34) apresenta um modelo de arquitetura genérico de bancos de dados, que nem sempre podem se encaixar nesse perfil, mas, afirma que a grande maioria são estruturados dessa forma, divididos em três níveis.

> O nível externo é o nível do usuário individual. Um usuário qualquer pode ser ou programador de aplicações ou um usuário final com qualquer grau de sofisticação.
>
> Normalmente, a visão que esse usuário tem dessa parte será um tanto abstrata quando comparada com o modo como os dados estão fisicamente armazenados. O termo ANSI/SPARC para a visão de um usuário individual é a visão externa. Uma visão externa é, portanto, o conteúdo do banco de dados visto por determinado usuário; em outras palavras, para esse usuário, a visão externa é o banco de dados.
>
> Cada visão externa é definida por meio de um esquema externo, o qual consiste basicamente em definições de cada um dos diversos tipos de registros externos naquela visão externa.
>
> A visão conceitual é uma representação de todo o conteúdo de informações do banco de dados, mais uma vez em uma ferramenta um tanto abstrata, em comparação com o modo como os dados são armazenados fisicamente. Em geral, ele também será bastante diferente do modo como os dados são visualizados por qualquer usuário em particular. Em termos gerais, a visão conceitual pretende ser uma visão dos dados “como eles realmente são”, em vez de forçar os usuários a vê-los pelas limitações (por exemplo) da linguagem ou do hardware que eles possam estar utilizando.
>
> A visão conceitual consiste em muitas ocorrências de cada um dos vários tipos de registros conceituais. Por exemplo, ela pode consistir em uma coleção de registros de departamentos, junto com uma coleção de ocorrências de registros de empregados, junto com uma coleção de ocorrências de registros de fornecedores, mais uma coleção de ocorrências de registros de peças, e assim por diante. Um registro conceitual não é necessariamente o mesmo que um registro externo, nem o mesmo que um registro armazenado.
>
> [...] A visão interna é uma representação de baixo nível do banco de dados por inteiro; ela consiste em muitas ocorrências de cada um dos vários tipos de registros internos. Registro interno é o termo ANSI/SPARC que representa a construção que temos chamado de registro armazenado. Assim, a visão interna ainda está separada do nível físico, pois ela não lida com registros físicos – também chamados blocos ou páginas – nem com quaisquer considerações específicas de dispositivos, como tamanhos de cilindros ou trilhas. Em outras palavras, a visão interna pressupõe efetivamente um espaço de endereços linear infinito; os detalhes de como esse espaço de endereços é mapeado por meio de armazenamento físico são bastante específicos para cada sistema, e foram deliberadamente omitidos da arquitetura geral.

A primeira visão, a externa, é uma resenha de como os dados serão armazenados e quais dados serão armazenados pelo sistema, seja ela vista pelo programador do sistema, seja pelo usuário do sistema.

A visão conceitual nos sugere uma visão de todas as ocorrências que o banco de dados irá receber e processar. Nessa visão, a intenção é visualizar os dados do modo como eles realmente são, utilizando-se de dados autênticos de registros conceituais para formar assim a visão conceitual do banco de dados.

A visão interna do banco de dados é formada pela ocorrência dos dados armazenados no banco. Essa visão nos oferece o espaço físico linear em que os dados serão inseridos, de modo que possam ser endereçados infinitamente.

2.3 LINGUAGEM SQL

Em seu artigo, Takai, Italiano e Ferreira (2005, p. 67) descrevem a Linguagem SQL como

> [...] um padrão de linguagem de consulta comercial que usa uma combinação de construtores em Álgebra e Cálculo relacional e possui as seguintes partes:
> - Linguagem de definição de dados (DDL);
> - Linguagem interativa de manipulação de dados (DML);
> - Incorporação DML (Embedded SQL);
> - Definição de Visões;
> - Autorização;
> - Integridade;
> - Controle de Transações;

Dessa forma, a linguagem SQL é um padrão de comandos para realizar funções e manipular dados através das semânticas pré-estabelecidas.

Em banco de dados, existem quatro funções básicas para manipular os dados armazenados através da linguagem SQL. São eles: INSERT, DELETE, UPDATE e SELECT.

O comando INSERT tem como função adicionar novos registros ao banco de dados.

Sua semântica é escrita dessa forma:

INSERT into <tabela>

VALUES <lista com os valores dos registros>;

O comando DELETE tem como finalidade remover registros que já constem no banco de dados. Esse comando faz com que sejam excluídos registros especificados pelo gerenciador. A semântica a ser realizada deve ser:

DELETE FROM <tabela>

WHERE <condição>;

O comando *UPDATE* é utilizado para modificar e atualizar valores de registros já armazenados no banco. Sua semântica é utilizada desse modo:

UPDATE <tabela>

SET <lista com os valores a serem modificados e seus novos valores> WHERE <condição>;

O comando SELECT é utilizado para selecionar registros que satisfaçam uma condição previamente instruída ao sistema pelo gerenciador. A semântica para utilizar esse comando é composta dessa forma:

SELECT <registros a serem selecionados>

FROM <tabela>

WHERE <condição>;

Dentro do comando SELECT existe uma imensa quantidade de alternativas para serem usadas, de forma que tragam como resultado valores e registros muito mais elaborados e mais específicos, de acordo com o que o programa ou o administrador do banco necessite.

2.4 SISTEMAS GERENCIADORES DE BANCOS DE DADOS

Em seu livro, Date descreve os Sistemas Gerenciadores de Bancos de Dados (SGBD) como “[...] o software que trata de todo o acesso ao banco de dados.” (DATE, 2003, p. 37).

Dessa forma, SGBD’s são programas de computador que administram e gerenciam bancos de dados. Cada SGBD é especificamente elaborado para gerenciar e administrar um tipo de modelo de banco de dados , e por consequência disso, deve-se utilizar um SGBD compatível com o modelo de dados que sua empresa ou aplicação necessita.

> Conceitualmente, o que ocorre é o seguinte:
> 1. Um usuário faz um pedido de acesso usando uma determinada sublinguagem de dados.
> 2. O SGBD intercepta o pedido e o analisa.
> 3. O SGBD, por sua vez, inspeciona o esquema externo para esse usuário, o mapeamento externo/conceitual correspondente, o esquema conceitual, o mapeamento conceitual/interno e a definição do banco de dados armazenado.

> 4. O SGBD executa as operações necessárias sobre o banco de dados armazenado. (DATE, 2003, p. 37)

O que Date nos apresenta é a maneira que o SGBD recebe, identifica por meio da SQL ou outra sublinguagem todos os campos e registros que sofrerão alguma alteração, e realiza suas tarefas e operações internamente.

Utilizando como exemplo um comando SELECT, o usuário insere sua requisição e a executa. O SGBD identifica os registros que serão o resultado, as tabelas de onde serão extraídas todas as informações necessárias, e as condições que o usuário informou. Agora, o SGBD analisa todo o banco de dados e suas esquematizações para encontrar internamente suas definições e localizações. Após todas essas etapas realizadas, ele retornará o resultado para o usuário caso todas as especificações estejam corretas.

Dessa forma, fica claro que os SGBD's nada mais são do que a interface entre usuário e o nível interno do banco de dados. É a ponte para as interações entre o sistema, o banco de dados, e o usuário.

Entretanto, para realizar todas essas requisições, o SGBD deve conter em sua composição, algumas funções, tais quais:

Definição de dados: de forma geral, o SGBD deve ser capaz de transformar os tipos de dados para poder entendê-los, de forma que, informado um código fonte, ele possa moldá-lo em código utilizável e concebível por ele;

Manipulação de dados: o SGDB deve ser capaz de realizar todas as tarefas básicas de banco de dados, como inserir novos registros, removê-los, atualizá-los e selecioná-los;

Otimização e execução: as requisições feitas pelo sistema ou pelo usuário devem passar por uma otimização, ou seja, o SGBD determinará um meio mais eficiente para se chegar ao resultado desejado;

Segurança e integridade de dados: o SGBD deve ter algum dispositivo ou meio de identificar requisições que violem suas definições de segurança, de modo que não exponha ou viole seus dados;

Recuperação de dados e concorrência: o SGBD ou outro *software* específico deve realizar um *backup* de informações periodicamente para que não haja perdas de dados e informações cruciais armazenadas;

Dicionário de dados: o SGBD deve conter em sua composição um banco de dados isolado, que forneça dados sobre o sistema. Em essência, um banco de dados do banco de dados, que contém os metadados, ou seja, informações dos dados armazenados e objetos;

Desempenho: o SGBD deve realizar todas essas funções e requisições do modo mais eficiente possível, no menor tempo possível.

No próximo capítulo, será abordado o modelo relacional de banco de dados, visto que, de forma geral, é o modelo mais utilizado hoje em dia e em muitos aspectos, o modelo com mais diferença em relação ao não-relacional. Portanto, para entendermos de uma maneira mais simples e mais esclarecedora o modelo não-relacional, deve-se antes, esclarecer e explicar o modelo relacional de banco de dados.

3 O MODELO RELACIONAL DE BANCO DE DADOS

O modelo relacional é a representação de um banco de dados, fundamentalmente constituído de ligações e relacionamentos entre entidades e tabelas.

3.1 INTRODUÇÃO

Macário e Baldo (2005, p. 1) apresentam-nos uma breve introdução ao modelo relacional da seguinte forma:

> O modelo relacional foi proposto por Edgar Codd em 1970, como uma nova maneira de representação de dados. Neste seu trabalho Codd mostrou que uma visão relacional dos dados permite a sua descrição em uma maneira natural, sem que sejam necessárias estruturas adicionais para sua representação, provendo uma maior independência dos dados em relação aos programas. Em complementação, apresentou bases para tratar problemas como redundância e consistência. Mais tarde, em outro trabalho, Codd definiu uma álgebra relacional e provou, por meio de sua equivalência com o cálculo relacional, que ela era relacionalmente completa, dando fundamentação teórica ao modelo relacional.

Já Date (2003, p. 50-51), em sua obra, descreve os três aspectos do modelo relacional.

> Aspecto estrutural: os dados no banco de dados são percebidos pelo usuário como tabelas, e nada além de tabelas.
> Aspecto de integridade: essas tabelas satisfazem a certas restrições de integritade [...]
> Aspecto manipulador: os operadores disponíveis para que o usuário possa manipular essas tabelas – por exemplo, para propósitos de busca de dados – são operadores que derivam tabelas a partir de outras tabelas. [...]

Dessa forma, fica evidenciada a percepção do usuário do banco como tabelas e, somente tabelas. E entre essas tabelas, existem os relacionamentos de cada uma delas, conectando-as umas às outras de forma relacionalmente estruturada.

Atualmente, esse modelo estrutural é o mais utilizado dentre todos os outros, por ser simples, representar integralmente todos os dados e trazer a descomplexidade de se realizar consultas internas.

Dadas essas comprovações, o mercado desse modelo cresceu e atualmente existem vários fornecedores de SGBD’s com o modelo relacional, dentre eles estão

IBM, Microsoft, Oracle. Hoje existem as alternativas de *softwares* livres de código aberto, facilitando o acesso às ferramentas disponíveis, entre eles, compõem o FireBird, MySQL e PostGres.

Date (2003, p. 54) esclarece que “o modelo relacional é muito mais que apenas ‘tabelas com restrições, projeções e junções’, embora muitas vezes seja caracterizado informalmente dessa maneira.”

A linguagem SQL foi originalmente desenvolvida pela IBM, e, conforme os aprimoramentos nessa linguagem, foi adotada de forma padrão para todos os programas gerenciadores de banco de dados do modelo relacional.

Sua criação dá-se em 1986, através da American National Standards Institute (ANSI) e foi nomeada SQL-86. Em 1989 foi revisada e fora incorporado restrições de integridade, tendo seu nome modificado para SQL-89. Novamente, em 1992, a American National Standards Institute, em conjunto com a International Standards Organization (ISO) lançou uma nova revisão, dessa vez muito mais expressiva e que alguns sistemas ainda utilizam. Em 1999, foram adicionadas mais algumas funções e algumas outras extensões, sendo denominada SQL:1999. Esta é, hoje em dia, a versão mais utilizada entre todas. Entretanto, em 2003 foi lançada uma nova versão, a SQL:2003, também conhecida como SQL:200n.

3.2 O MODELO RELACIONAL

O modelo relacional é, fundamentalmente, constituído pelo conjunto de relações, a qual pode ser única ou múltipla, com nomes definidos de forma diferente, sem que haja duplicidade. O esquema do modelo relacional é o conjunto de esquemas de cada uma das relações consistidas dentro do banco de dados.

No modelo relacional, o principal fundamento de sua constuição é denominado relação. Uma relação é consistida de dois elementos: esquema e instância. Macário e Baldo (2005, p. 3) explicam esses dois conceitos:

> o esquema especifica o nome da relação e o nome e o domínio de cada coluna, também denominada atributo ou campo da relação.
> [...] A instância de uma relação é o conjunto de linhas, também denominadas tuplas ou registros, distintas entre si, que compõem a relação de um dado momento. Ela é variável, já que o número de tuplas e o conteúdo de seus atributos podem variar ao longo do tempo. A instância de uma relação deve seguir sempre o seu respectivo esquema, respeitando o número de atributos definidos, bem como seus domínios.

Dessa maneira, percebemos que o esquema identifica a relação e também o atributo. Um exemplo de esquema de relação pode ser definido dessa maneira:

Pessoa (id: string, nome: string, idade: integer, cpf: string, nascimento: date).

Nesse exemplo, está definido a relação com o nome Pessoa, com os atributos id, nome, idade, cpf e nascimento, cujos domínios estão claramente especificados como: *string, string, integer, string e date.*

O modelo relacional apenas considera relações que venham a respeitar os números de atributos e domínios definidos. Na tabela abaixo podemos verificar um exemplo de instância para a tabela Pessoa:

Tabela 1 – Exemplo de instância para a tabela Pessoa.

ID	Nome	Idade	CPF	Nascimento
1	João	32	111.222.333-44	01/01/1982
2	Ricardo	25	222.333.444-55	02/02/1989
3	Ana	21	333.444.555-66	03/03/1993
4	Jonas	30	444.555.666-77	04/04/1984

Fonte: Próprio Autor

O número de registros contidos dentro da tabela é denominado cardinalidade, e o número de seus atributos leva a denominação de grau. Note que ma tabela apresentada acima, a cardinalidade da tabela é 4, e seu grau é 5.

3.2.1 Utilizando SQL para Criação e Modificação de Tabelas

Segundo a linguagem SQL, a palavra utilizada para definir e identificar uma relação é denominada TABLE. Dentro da linguagem SQL, ainda há uma sublinguagem que compreende os comandos básicos de banco de dados, como a criação, remoção, atualização e modificação das tabelas existentes. Essa sublinguagem é conhecida como *Data Definition Language* (DDL), ou, em português, Linguagem de Definição de Dados.

Utilizando-se da linguagem SQL, a semântica para criação de tabelas, seguindo o exemplo dado na seção anterior da tabela Pessoa, dá-se desta forma:

CREATE TABLE Pessoa (id: char(15), nome: char(40), idade: integer(3), cpf: char(14), nascimento: date);

É importante esclarecer que não estamos definindo ainda nenhum valor aos atributos, somente estamos criando a relação, para que mais pra frente possamos populá-los com as respectivas informações.

O comando utilizado pela DDL para a remoção de tabelas é o DROP, dessa forma, ainda seguindo com o exemplo da tabela Pessoa, temos:

DROP TABLE Pessoa;

Note aqui que, ao utilizar esse comando, removemos tudo relacionado à tabela pessoa, e todos os registros contidos dentro da tabela serão removidos juntamente com a sua relação.

Para podermos modificar ou alterar uma tabela dentro do sistemas, utilizamos o comando ALTER TABLE. Utilizando ainda a tabela Pessoa, alteraremos sua composição para adicionarmos um atributo denominado sexo, sendo seu domínio *character.* Dessa forma, temos:

ALTER TABLE Pessoa ADD COLUMN sexo: char(9);

Com esse comando, estamos alterando o esquema e adicionando para cada registro atual, um novo atributo com o nome sexo. Entretanto, esse valor será integrado com o valor *null*, visto que ainda não inserimos nenhum valor à ele.

A DDL ainda possui subconjuntos para a modificação e recuperação de dados existentes nas tabelas.

Utilizamos o comando INSERT INTO para a adição de novos registros nas relações existentes no banco de dados. Dessa forma, temos:

INSERT INTO Pessoa (id, nome, idade, cpf, nascimento)

VALUES (1, ‘João’, ’32’, ‘111.222.333-44’, ‘1982-01-01’);

Note aqui que, na linguagem DDL, o padrão para inserção de data é diferente do europeu, utilizado no Brasil. Nesse caso, inserimos primeiro o ano, seguido do mês e do dia. Nesse exemplo, tudo o que está compreendido entre o primeiro parênteses de VALUES, até o final de seu parênteses, serão inseridos na relação Pessoa. A vírgula nesse caso serve para a identificação de cada atributo e a sua atribuição de valores, para que não haja erros na hora de inserir os dados.

No modelo relacional de banco de dados, a atomicidade é um dos itens mais importantes a se notar. A atomicidade garante que, caso haja qualquer erro na entrada de dados, nenhum valor será adicionado, mesmo que apenas um atributo esteja com erro. A atomicidade não se aplica para erros gerados pelo usuário quanto à informação apresentada, apenas para erros de semântica, como por exemplo na inserção de

dados acima, se a data fosse informada da maneira padrão, ou seja, 01/01/1982. O sistema gerenciador iria negar a requisição e nenhum dado seria inserido à tabela.

Para a remoção de registros das tabelas, é utilizado o comando DELETE FROM. Pode-se remover todos os registros existentes na tabela, ou apenas remover dados específicos, conforme uma condição informada ao sistema. No exemplo, removeremos todos os registros que contenham no atributo nome, o valor ‘João’. Temos, então:

```
DELETE FROM Pessoa P
WHERE P.nome=’João’;
```

Dessa forma, removeremos todos os registros que contenham o valor João no nome, mesmo que haja no sistema mais de um registro com este valor.

A alteração de valores de atributos da tabela é gerenciada pelo comando UPDATE FROM. Parecida com o comando DELETE, o UPDATE pode gerenciar desde apenas um registro até vários em apenas um comando. Temos, então:

```
UPDATE FROM Pessoa S
SET S.idade=idade+1
WHERE S.cpf=’111.222.333-44’
```

Dessa forma, alteramos apenas um registro, incrementando a idade para um ano a mais, cujo CPF seja 111.222.333-44.

Em outro exemplo, alteraremos todas as idades, adicionando 2 anos, para todos os registros que tiverem seu atributo idade maior que 20.

```
UPDATE FROM Pessoa S
SET S.idade=idade+2
WHERE S.idade >= 20
```

3.2.2 Restrições

A melhor forma do SGBD garantir que as informações nele inseridas são de caráter verídico e de qualidade, é evitando a entrada de informações inconsistentes e incorretas, através de restrições gerenciadas pelo sistema.

Uma das restrições é a restrição de integridade, a qual Macário e Baldo (2005, p. 5) definem como “*uma condição especificada no esquema da base de dados para restringir a informação a ser armazenada [...]”.* Ou seja, todas as informações a serem inseridas no banco de dados passarão por uma condição para validá-la. Caso a

informação satisfaça todas as restrições definidas, ela é considerada verdadeira e é inserida ao banco de dados. Caso contrário, ela é descartada.

Várias restrições de integridade podem ser definidades dentro de um banco de dados, para que haja um maior controle da entrada de dados no sistema.

As restrições podem ser especificadas e conferidas em dois momentos distintos:

Especificação da restrição: ocorre na definição do esquema do banco de dados e é gerenciado pelo usuário ou pelo administrador do banco de dados;

- Conferência da restrição: o próprio sistema realiza a tarefa toda vez que algum registro ou relação é alterado por outro sistema externo a esse.
- Outra restrição existente é a restrição de chaves, utilizada para que os registros da tabela sejam únicos e exclusivos. Para essa restrição, é necessário identificar um conjunto mínimo de atributos, e que devem ser distintos de todos os registros de uma tabela.

O conjunto de atributos é denominado chave candidata, e como as outras, deve atender alguns requisitos, como:

- Não pode existir mais de um registro com o mesmo valor para um atributo, de forma que essa chave identifica distintamente qualquer registro da tabela válida;
- Na remoção de qualquer atributo consistido dentro da chave, esta, deixa de identificar unicamente os registros.

No caso da segunda restrição ser violada, é denominado que a chave candidata é uma super-chave. Como exemplo, temos a tabela Pessoa. Id é uma chave que identifica uma pessoa. Utilizando o conjunto {id e cpf}, temos uma super-chave da tabela, visto que, se removermos o atributo cpf, ainda assim a id continua identificando unicamente o registro. Todos os atributos em conjunto de uma tabela formam uma super-chave.

Para relações, sempre deve existir uma chave, denominada de chave primária. Cabe ao administrador ou programador definir qual será a melhor opção para se tornar chave primária, sendo ela unicamente definida na tabela. Se definíssemos o atributo idade como chave primária da tabela, teríamos que apenas uma pessoa poderia ter 20 anos de idade, e nenhuma outra poderia ter esse valor. Em hipótese alguma a chave primária poderá ter o valor como *null.* Deve-se, sempre, ter um valor atribuído à ela.

Os comandos UNIQUE e PRIMARY KEY identificam, respectivamente, a chave a chave primária da tabela.

Utilizando do exemplo Pessoa, definiríamos como chave primária o atributo id. Dessa forma, temos, na criação da tabela:

CREATE TABLE Pessoa (id: char(15), nome: char(40), idade: integer(3), cpf: char(14), nascimento: date, PRIMARY KEY (id));

No desejo da obtenção de duas chaves, composta pelos atributos ID e CPF, de forma que ainda seja identificada unicamente pelo ID, teríamos:

CREATE TABLE Pessoa (id: char(15), nome: char(40), idade: integer(3), cpf: char(14), nascimento: date,

UNIQUE (id, cpf), CONSTRAINT PessoaPrimary PRIMARY KEY (id));

A utilização da *constraint,* nesse caso, tem como objetivo identificar a chave primária, facilitando sua utilização caso haja violação na sua restrição.

Ainda, nota-se que no exemplo temos que cada pessoa possui apenas um número de CPF atrelado a ela. O que nesse caso é real, mas que, utilizadas de formas equivocadas, podem restringir campos que serão necessários às práticas do cotidiano do banco de dados, como por exemplo, restringir salários de colaboradores de uma empresa, de forma que dois funcionários distintos não possam eventualmente receber o mesmo salário.

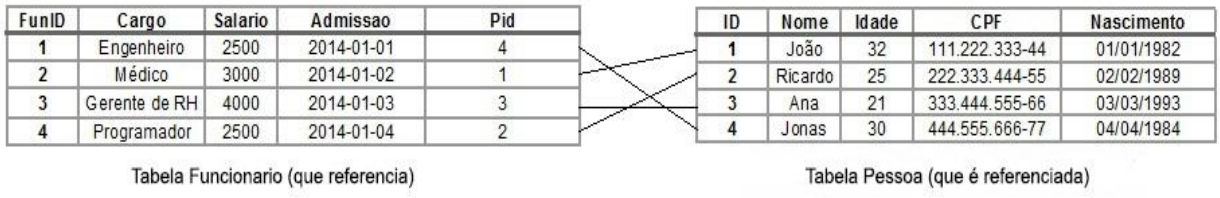
Neste modelo relacional, é comum termos uma informação exterior, oriunda de uma outra tabela existente no banco de dados. Caso haja a modificação de uma informação, a outra também deve ser verificada e modificada se for o caso, de forma a garantir a consistência dos dados. A restrição existente para isso é denominada chave estrangeira, e realiza a checagem da existência desse atributo dentro das duas tabelas relacionadas.

Para realizar essa relação, criaremos aqui uma tabela chamada Funcionário, com os atributos Funid, cargo, salario, admissao e pid.

CREATE TABLE Funcionario (Funid: char(20), cargo: char(20), salario: double(10), admissao: date, pid: char(15), PRIMARY KEY (Funid)).

Dessa forma, geramos a seguinte relação:

Figura 1. Exemplo de chave estrangeira



FunID	Cargo	Salario	Admissao	Pid
1	Engenheiro	2500	2014-01-01	4
2	Médico	3000	2014-01-02	1
3	Gerente de RH	4000	2014-01-03	3
4	Programador	2500	2014-01-04	2

Tabela Funcionario (que referencia)

ID	Nome	Idade	CPF	Nascimento
1	João	32	111.222.333-44	01/01/1982
2	Ricardo	25	222.333.444-55	02/02/1989
3	Ana	21	333.444.555-66	03/03/1993
4	Jonas	30	444.555.666-77	04/04/1984

Tabela Pessoa (que é referenciada)

Fonte: Próprio autor

Note aqui, que o atributo Pid é uma chave estrangeira, proveniente da tabela Pessoa, cujo atributo referenciado é o ID. Dessa maneira, se tentarmos adicionar um novo registro na tabela Funcionário, especificando o Pid 5, o sistema rejeitará a requisição, visto que não existe nenhum registro na tabela Pessoa com ID 5. Da mesma maneira, se tentarmos remover qualquer registro da tabela Pessoa nesse momento, o sistema negará, e acusará que há um registro na tabela Funcionário que referencia o registro a ser removido. Para que a remoção seja válida, é preciso que o registro a ser removido não haja nenhuma referência a ela, ou seja, deve-se remover todos os registros de outras tabelas, nesse caso da tabela Funcionário, que fazem referência a ele, e assim, o sistema aceitará a remoção do registro.

Em SQL, o comando utilizado para especificar uma chave estrangeira é denominado FOREIGN KEY, e para o exemplo da figura acima, utiliza-se:

CREATE TABLE Funcionario (Funid: char(20), cargo: char(20), salario: double(10), admissao: date, pid: char(15),

PRIMARY KEY (Funid),

FOREIGN KEY (Pid) REFERENCES Pessoa).

Por meio desta execução, foi definido que somente as pessoas cadastradas na tabela Pessoa podem ser registradas na tabela Funcionários.

A linguagem SQL ainda nos fornece algumas soluções para garantir a restrição de integridade. O comando NO ACTION faz com que o banco de dados ignore a execução de uma função. O comando CASCADE é utilizado para que, na remoção de um registro que esteja relacionado ou sendo referenciado, todos esses registros serão

removidos juntamente. E os comandos SET NULL/ SET DEFAULT são utilizados para atribuir valores para a chave estrangeira do registro que está sendo referenciado.

Cogitando que, na hipótese do banco de dador vir a excluir um registro, ele deverá utilizar o comando CASCADE para remover juntamente todas suas relações e referências, e , no caso de modificação e atualização de qualquer registro, o sistema descarte o comando, então temos:

CREATE TABLE Funcionario (Funid: char(20), cargo: char(20), salario: double(10), admissao: date, pid: char(15),

PRIMARY KEY (Funid),

FOREIGN KEY (Pid) REFERENCES Pessoa

ON DELETE CASCADE

ON UPDATE NO ACTION).

Para a realização de consultas e seleção de conteúdo de tabelas pelo banco de dados, o comando utilizado é o SELECT FROM. O resultado desse comando, é uma nova tabela formado pelas tabelas e registros consultados. Como exemplo, utilizaremos a tabela Pessoa, e apenas selecionaremos pessoas que tenham mais de 25 anos de idade. Temos:

SELECT * FROM Pessoa P

WHERE P.idade >= 25.

A figura 2 nos mostra a resposta dessa requisição ao banco de dados.

Figura 2 – Exemplo de pesquisa na tabela Pessoa

ID	Nome	Idade	CPF	Nascimento
1	João	32	111.222.333-44	01/01/1982
2	Ricardo	25	222.333.444-55	02/02/1989
4	Jonas	30	444.555.666-77	04/04/1984

Fonte: Próprio autor

Utilizando o asterisco desta forma, estamos solicitando ao banco de dados que traga na pesquisa todos os atributos da tabela solicitada. Para que ele resulte somente atributos específicos, substituímos na requisição o * pelos atributos que necessitamos. Então, utilizamos desta maneira:

SELECT id, nome, idade FROM Pessoa P

WHERE P.idade >= 25.

Dessa forma, solicitamos ao banco de dados que sejam exibidos apenas os atributos id, nome e idade, das pessoas que estiverem cadastradas e com idade igual ou superior a 25 anos. Na figura 3, Podemos ilustrar mais claramente o resultado dessa requisição.

Figura 3 – Exemplo de pesquisa com especificação de atributos

ID	Nome	Idade
1	João	32
2	Ricardo	25
4	Jonas	30

Fonte: Próprio autor

A partir dessas premissas, informações e análise do modelo relacional, podemos focar no modelo não-relacional de banco de dados. Veremos uma introdução sobre o modelo, uma análise dos modelos e SGBD’s existentes dentro do NoSQL, a utilização de alguns deles no cotidiano, o elevado nível de escalabilidade dentro do NoSQL, suas funções e seus esquemas internos, quando disponíveis.

4 O BANCO DE DADOS NÃO-RELACIONAL E A ESCALABILIDADE

Todos os fundamentos abordados dentro do modelo relacional, agora serão ignorados. O modelo não-relacional, como o próprio nome diz, tem como principal característica a inexistência de relações entre os dados em si armazenados. Veremos, nesse capítulo, desde o básico do NoSQL, como uma introdução à ele, seus conceitos e definições, até uma comparação analítica entre o modelo relacional e não-relacional, de forma a apontar as vantagens e desvantagens de ambos.

Ainda, será abordado todo o conceito de escalabilidade de banco de dados, e porquê e como o modelo não-relacional trata desse tema muito mais facilmente que o modelo relacional de banco de dados.

4.1 INTRODUÇÃO

O modelo relacional de banco de dados foi concebido primordialmente pela necessidade de um modelo que suportasse uma grande quantidade de dados. Com a evolução dos sistemas, essa necessidade foi se acentuando cada vez mais, visto que o modelo relacional de banco de dados tem um limite de dados suportados.

Além desse suporte a grande quantidades de dados, o NoSQL também economiza o tempo que sua empresa gastaria, por exemplo, criando esquemas e relações entre tabelas, enquanto o modelo não-relacional permite que os esquemas sejam personalizados conforme a necessidade do aplicativo.

O surgimento do termo “NoSQL” só foi acontecer em 1990, por Carlos Strozzi, criador de um banco de dados relacional de código aberto, denominado Strozzi NoSQL. Sadalage e Fowler (2013, p. 34) explicam que “o nome vem do fato que o banco de dados não utiliza SQL como uma linguagem de consulta. Em vez disso, ele é manipulado por meio de shell scripts. [...]”. Contudo, a denominação utilizada por Strozzi do NoSQL naquela época, não tem quaisquer identificações com o modelo não-relacional que temos hoje.

O modelo não-relacional que conhecemos hoje tem seu surgimento em uma reunião datada de 11 de junho de 2009, em São Francisco, nos Estados Unidos. Nessa reunião, o então organizador do evento Johan Oskarsson, desenvolvedor de software de Londres, estava entusiasmado com essa nova ideia de bancos de dados.

Para que a reunião fosse nomeada, ele solicitou em seu canal #cassandra do IRC, uma das sugestões dadas fora o termo “NoSQL” de Eric Evans, ainda que, naquele tempo, não era de se esperar que esse termo fosse um dia nomear, segundo Sadalage e Fowler (2013, p. 35), uma “[...] tendência tecnológica como um todo.”

Os bancos de dados NoSQL, como o próprio nome diz, não utilizam-se da linguagem SQL, em vez disso, fazem uso de sublinguagens de consulta, que, por facilidade de aprendizado, são muito semelhantes ao SQL. Ainda, uma de suas características é que, em sua imensa maioria, os SGBD’s desse modelo são de código aberto, embora possam ser implementados em sistemas de código fechado.

Outra característica do NoSQL é sua alternativa de poder ter seu processamento distribuído entre *clusters*, ao passo que o modelo relacional de banco de dados continua tendo muita dificuldade para lidar com isso. Não são todos os modelos dentro do NoSQL que utilizam dessa ferramenta. O banco de dados de grafos do não-relacional faz uso de um modelo de distribuição de processamento parecido com o que temos em bancos de dados relacionais.

O modelo não-relacional de banco de dados permitiu às empresas que fizessem uso de mais de um modelo de armazenamento de dados. Tal prática é conhecida como persistência poliglota e permite às empresas fazer uso, fundamentada em uma análise, do melhor modelo de banco de dados disponível conforme as suas necessidades.

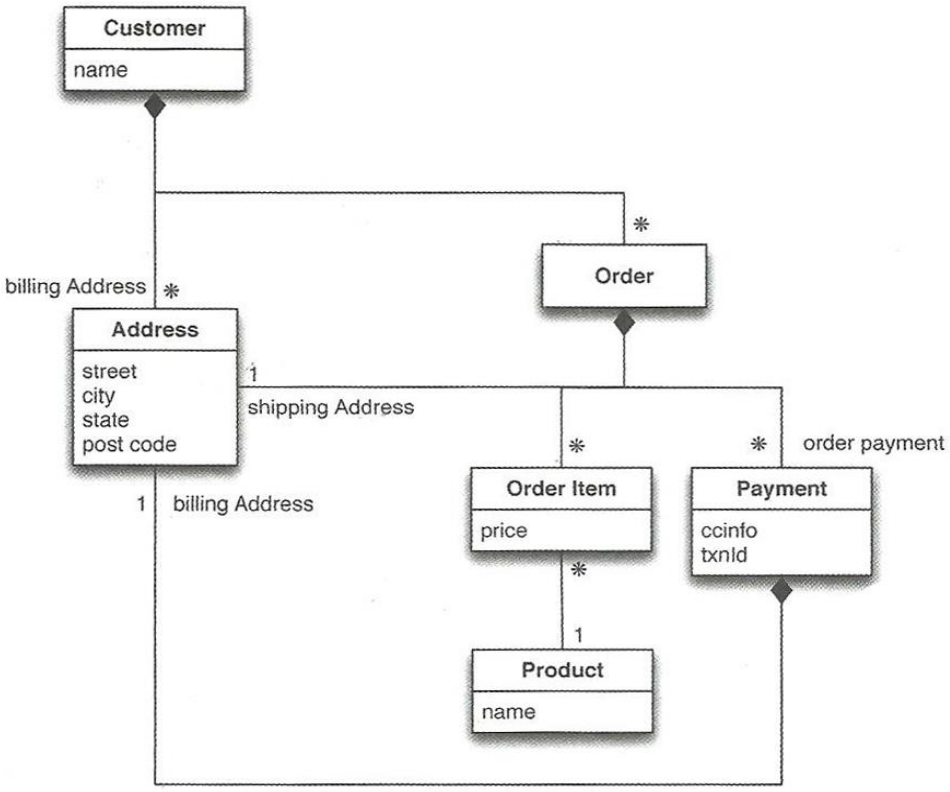
Dentro deste modelo, temos vários SGBD’s de acordo com sua modelagem de dados. Falaremos de todos, de forma a obter um maior conhecimento sobre todas as funcionalidades do modelo não-relacional. Entre esses modelos de dados, temos o banco de dados de chave-valor, o banco de dados de documentos, o armazenamento em famílias de colunas e os bancos de dados de grafos. Ainda, dentro desses modelos, podem existir SGBD’s capazes de trabalhar em dois modelos distintos, como o caso do OrientDB, que pode ser utilizado tanto como banco de dados de documentos quanto banco de dados de grafos.

4.2 MODELOS DE DADOS AGREGADOS

A orientação agregada de dados é utilizada de forma diferente da que estamos acostumados com o modelo relacional. Esta, agora, pressupõe que você utilizará dados na forma de unidades que sejam mais complexas que um mero conjunto de

registros. Os modelos que utilizam desta ferramenta são os do tipo de chave-valor, de documento e de família de colunas. Um agregado é “um conjunto de objetos relacionados que desejamos tratar como uma unidade.” (SADALAGE; FOWLER, 2013, p. 42). Dessa forma, temos que a estruturação dos dados nesse modelo se dará de forma que os agregados principais ficarão acima e, os agregados seguintes a esses, que fazem referência aos principais, ficarão abaixo destes. Entre os agregados principais, eles serão representados lado a lado. Como exemplo, utilizaremos um exemplo de um website de comércio eletrônico. Dessa forma, temos o seguinte modelo de dados agregados:

Figura 4 – Exemplo de modelagem de dados agregados para website de comércio eletrônico



Fonte: SADALAGE; FOWLER, 2013, p. 47

Podemos dizer, segundo este exemplo, que existem dois grandes agregados, aqui representados por *Customer* (Cliente) e *Order* (Pedido). No agregado Cliente, temos uma lista de endereços de cobrança; o pedido incorpora uma lista de itens, o qual também incorpora uma lista de produtos; ainda, o pedido contém o endereço de envio e o endereço de pagamento. É possível notar também nesse exemplo que os subitens dos agregados principais ficam exatamente abaixo desses.

Para Sadalage e Fowler (2013, p. 49), “o motivo mais importante para a orientação de agregados é que ela realmente auxilia a execução em um cluster, que é o argumento crucial para a ascensão do NoSQL.” Dessa forma, é enfatizada novamente a importância da contribuição do NoSQL para a solução de problemas de escalabilidade.

4.3 O MODELO DE DADOS DE CHAVE-VALOR

Bancos de dados de chave-valor são enraizados no modelo de dados agregados. Contudo, o agregado, nesse caso, é opaco para o banco de dados e, consequentemente, pode-se armazenar o que você bem entender no agregado.

Sadalage e Fowler definem o modelo de dados de chave-valor como “[...] uma tabela hash simples, utilizada principalmente quando todo o acesso ao banco de dados é feito por meio da chave primária.” (2013, p. 123). Em um banco de dados do modelo relacional, em termos comparativos, temos dois atributos, sendo eles ID e NAME, os quais são, respectivamente, chave primária e o outro armazena dados do tipo *string.* Enquanto isso, no modelo de chave-valor, as chaves não se importarão com seu conteúdo, cabe ao aplicativo entender o que fora armazenado dentro delas.

Neste modelo, o usuário tem as funções para selecionar os valores de determinadas chaves, inserir novos valores para chaves existentes e deletar chaves também existentes. Sadalage e Fowler novamente deixam claro que esse tipo de modelo tem uma boa escalabilidade, de forma que “[...] depósitos de chave-valor sempre fazem o acesso pela chave primária, eles têm, geralmente, um ótimo desempenho e podem ser escaláveis facilmente.” (2013, p.123).

Existem vários SGBD’s disponíveis para esse modelo, os mais utilizados são o Riak, Redis, Memcached DB, Berkeley DB, HamsterDB, Amazon DynamoDB e Project Voldemort. No SGBD Redis, o agregado pode ter qualquer estrutura de dados, e ainda tem suporte ao armazenamento de listas, conjuntos, *hashes* e pode fazer operações de intervalos, diferença, união e intersecção. Isso o diferencia dos demais SGBD’s utilizados para este modelo, e permite que seja utilizado de forma mais abrangente no contexto do modelo de chave-valor.

Temos dois modos de armazenar dados em um banco de dados de chave-valor. Podemos criar *buckets* e armazenar todas as informações em um mesmo *bucket* e assim, utilizar uma única chave e um valor único para todos os objetos dele.

A outra forma de se armazenar dados, é separando os objetos em vários *buckets* e, dessa forma, segmentá-los. Como consequência, apenas os *buckets* necessários à aplicação poderiam ser lidos, sem alteração de chaves.

Para a criação de *buckets* nesse modelo, mais especificamente do SGBD Riak, é utilizado o comando CREATEBUCKET(BUCKETNAME). Dessa forma, temos:

```
Bucket bucket = connection
.createBucket(bucketName)
.execute();
```

Uma vez que todos os valores tenham sido repassados para outros nodos, o Riak têm uma característica denominada consistência eventual. Ela é responsável pelas maneiras de se resolver conflitos e disponibiliza ao Riak duas possibilidades: ou a gravação mais recente de dados prevalece sobre as mais antigas, ou todos os valores são devolvidos para o cliente, o qual deverá resolvê-lo da melhor forma possível.

O SGBD Riak é baseado em um ambiente HTTP, e com isso, todas as operações necessárias podem ser realizadas através de um navegador web. O comando padrão para inserir dados e coletá-los é definido como *curl*. Temos então que, para todas as operações posteriores, deve-se usar o *curl* imediatamente à frente, e após isso o comando desejado. O comando POST é utilizado para gravar os dados, armazenando no *bucket* criado. Desse modo, temos:

```
curl –v –X POST –d ‘
{
[dados a serem gravados no banco]
}’
http://localhost:8098/buckets/session/keys/[chave]
```

Nesse contexto, após /sessions/keys/ deveria ser criado a chave para se armazenar os dados. Essa chave é definida pelo cliente, que deve pensar na melhor forma de se gerá-la.

Para que se possa recuperar esses dados inseridos na chave, utiliza-se o comando:

```
curl –i http://localhost:8098/buckets/session/keys/[chave]
```

Desta forma, o banco de dados vai realizar a pesquisa baseada na chave criada previamente no comando POST.

Em termos de escalabilidade, esse modelo de banco de dados não-relacional permite a utilização de um fundamento chamado fragmentação. Esta, permite que o cliente possa escalar horizontalmente o banco de dados, de forma que possa armazenar os dados em diferentes servidores espalhados pelo globo, contanto que trabalhem simultaneamente uns com os outros. Cada gravação efetuada no banco de dados é replicada pelos nodos dos servidores, de forma a atualizar os dados nos outros servidores.

Dessa forma, não é necessário uma máquina com uma alta escalabilidade para que se possa utilizar este modelo. O próprio modelo dispõe de um mecanismo que o torna extremamente escalável, ainda que o utilizador deste tenha que armazenar uma grande quantidade de dados e realizar operações que poderiam sobrecarregar SGBD's do modelo relacional.

Ainda assim, deve-se ter certa cautela para utilizar este modelo de dados. Ele é recomendado em casos que necessitam armazenar informações de sessão, visto que a chave pode ser utilizada como a *sessionid,* armazenando todas as informações dela e manipulando-a em uma única solicitação ou casos em que os usuários possuam apenas um *userid*, de modo que esta seja a chave e todas as informações relacionadas ao *userid* sejam armazenadas dentro desta chave.

Em outros casos, como nos casos de relacionamento de dados, em que você precise estabelecer relações ou correlações entre dados ou conjunto de chaves; ou no caso de transações com múltiplas operações, que manipulam e gravam várias chaves ao mesmo tempo, ou que haja necessidade de parar ou reverter a gravação dos dados; ou mesmo queira realizar consulta por dados e não pela chave, ou até mesmo parte dos dados armazenados; e até operações por conjunto, onde há necessidade de se trabalhar em várias chaves ao mesmo tempo. Nesses casos, o modelo de chave-valor não é a melhor opção.

4.4 BANCOS DE DADOS DE DOCUMENTOS

Esse modelo consiste em armazenar e recuperar documentos, os quais podem ser XML, JSON, BSON, entre outros. Sadalage e Fowler definem documentos como "[...] estruturas de dados na forma de árvores hierárquicas e autodescritivas, constituídas de mapas, coleções e valores escalares." (2013, p. 133). Os documentos

que forem armazenados podem ser semelhantes, mas não possuem exatamente os mesmos valores ou atribuições.

Os SGBD's deste modelo mais utilizados são o MongoDB, CouchDB, Terrastore, OrientDB, RavenDB e o Lotus Notes. Cada um desses possui características diferentes uns dos outros, mas, fundamentalmente todos se enraizam no modelo de banco de dados de documentos.

No MongoDB, existem várias semelhanças com sistemas gerenciadores de banco de dados relacionais, como por exemplo a instância. Uma instância no MongoDB é igual a uma instância no SGBD relacional. Outro exemplo são as tabelas, utilizadas pelo modelo relacional, que, no MongoDB equivalem às coleções. Em todo caso, sempre que formos armazenar um documento, é preciso especificar em qual banco de dados e coleção ele irá pertencer.

Se tratando do modelo de documentos, é possível garantir a atomicidade quando se modifica apenas um documento. Modificar mais de um documento no MongoDB não é possível. Um dos únicos a oferecer esse tipo de solução é o RavenDB, que, de acordo com Sadalage e Fowler "[...] suportam transações com múltiplas operações." (2013, p. 137).

A utilização da configuração mestre-escravo por este modelo tem como objetivo melhorar a disponibilidade. Esta configuração permite que os dados fiquem sempre disponíveis em nodos secundários, mesmo que o nodo mestre, ou nodo principal, fique indisponível. A funcionalidade pode ser observada desta maneira: os dados manipulados são enviados ao nodo mestre, e após isso, os dados são replicados para os nodos secundários. Todos os nodos podem realizar uma votação com o mesmo peso de votos, contanto que o cliente não tenha alterado isso para favorecer algum nodo em particular, o qual pode ter uma característica fundamental para todo o processo. Dessa forma, se a transmissão do mestre para os escravos falhar, estes realizarão uma votação para eleger o novo nodo mestre. As modificações posteriores a esta votação serão automaticamente direcionadas ao novo nodo mestre e assim, replicadas aos escravos. O nodo que era mestre e passou pelo *downtime*, assim ficando novamente disponível, volta ao conjunto de nodos como escravo e relaciona seus dados aos escravos para se atualizar.

Sadalage e Fowler (2013, p. 140) salientam a utilização do conjunto de réplicas para "[...] redundância de dados, recuperação automática de falhas, ampliação na

capacidade de leitura, manutenção de servidor sem tirar o aplicativo do ar e recuperação após desastres.”

Uma das diferenças entre esse modelo e o anterior, o de chave-valor, é a sua função de consulta. Agora, temos uma funcionalidade para consultar dados “[...] dentro do documento sem ter de recuperar o documento inteiro por sua chave e depois examiná-lo”. (SADALAGE; FOWLER, 2013, p. 140). Sendo assim, esse modelo se assemelha ao modelo de consulta dos bancos de dados relacionais.

Em uma estrutura SQL, podemos realizar consultas dessa forma:

SELECT * FROM pedidos

Já em linguagem JSON, disponível para o MongoDB, temos:

db.compra.find()

Utilizando parametrização, em SQL poderíamos consultar os pedidos feitos por um único id_cliente dessa forma:

SELECT * FROM pedidos

WHERE id_cliente = ‘112’

Em JSON, a semelhança é evidente:

db.compra.find({“id_cliente”:”112”})

Utilizando uma gama maior de especificações, poderíamos expressar em SQL a consulta de pedidos feitos por clientes, constando o termo NoSQL nos itens comprados, desta maneira:

SELECT * FROM CompraCliente, ItemCompra, produto

WHERE

CompraCliente.IdPedido = ItemCompra.CompraClienteID

AND ItemCompra.produtoID = produto.produtoID

AND produto.nome LIKE ‘%NoSQL%’

No MongoDB, a semântica aludida é utilizada desta maneira:

db.compras.find({“items.produto.nome”:/NoSQL/})

Em termos de escalabilidade, este modelo dispõe da possibilidade de se adicionar nodos escravos somente para a leitura, e assim direcionar somente a leitura para todos os nodos escravos. A dificuldade de adição de novos nodos é quase mínima, visto que ele pode ser adicionado a qualquer momento e, assim que incluído no novo conjunto de nodos, ele automaticamente sincronizará todos os dados com os outros nodos secundários. Este, também obtém o direito de voto caso o nodo principal não responda as solicitações, e pode ser alterada sua prioridade de voto.

Quanto à gravação, esse modelo também dispõe do método de fragmentação, em que, os dados podem ser armazenados em diferentes nodos. Essa divisão é sempre balanceada, para que nenhum nodo fique sobrecarregado. Nesse caso, não será necessário parar a aplicação para a adição de novos nodos de fragmentação de gravação, contudo, segundo Sadalage e Fowler, “[...] o cluster possa não ter desempenho ideal quando grandes quantidades de dados estiverem sendo movimentados para rebalancear os fragmentos.” (2013, p. 143) Ou seja, os dados serão automaticamente realocados e balanceados, mas, para isso, será necessário abdicar por um breve momento de seu desempenho e de sua disponibilidade.

Esse modelo de dados é recomendável para uso quando tiver a necessidade de registro de eventos (*log*), visto que, de acordo com Sadalage e Fowler (2013, p. 144), “[...] podem armazenar todos esses diferentes tipos de evento e atuar como um depósito central de dados para o registro desses eventos.”

Também pode ser utilizado nos casos de Sistema de Gerenciamento de Conteúdo ou plataformas de blog, pois são uma boa ferramenta para gerenciar os comentários feitos por usuários, registros, perfis e documentos visualizados por estes através da *web.*

Em casos de análises *web* ou em tempo real, “uma vez que partes do documento podem ser atualizadas, é muito fácil armazenar visualizações de páginas ou visitantes únicos.” (SADALAGE; FOWLER, 2013, p. 145). Aplicativos de comércio eletrônico também se beneficiam deste modelo, pela necessidade de se ter um esquema flexível, e capacidade de modificar suas estruturas de dados.

Nos casos de transações complexas com múltiplas operações e consultas em estruturas agregadas variáveis, este modelo não é a melhor opção. No primeiro caso, o modelo de documentos não garante atomicidade em múltiplos documentos, e no segundo caso, é extramemente complexo e trabalhoso ter que modificar os agregados toda hora que surgir a necessidade.

4.5 O MODELO DE ARMAZENAMENTO EM FAMÍLIAS DE COLUNAS

Este modelo consiste especialmente que se “[...] armazene dados com chaves mapeadas para valores, e os valores são agrupados em múltiplas famílias de colunas, cada uma dessas famílias de colunas funcionando como um mapa de dados.” (SADALAGE; FOWLER, 2013, p. 147).

Os SGBD's disponíveis para esse modelos são o Cassandra, o mais popular dentre todos, o HBase, Hypertable e Amazon DynamoDB. O Cassandra distribui suas operações pelo *cluster*, e não possui um nodo principal, possibilitando a qualquer um de seus nodos realizar tanto gravação, quanto leitura dos dados.

O conceito fundamental é a utilização de colunas para o armazenamento de dados. "Uma coluna no Cassandra consiste em um par de nome-valor em que o nome também se comporta como a chave." (SADALAGE; FOWLER, 2013, p. 148). Dessa maneira, cada par de chave-valor é uma coluna, e é sempre armazenada com um valor de timestamp. Esse valor de timestamp é tratado em casos de desatualização de dados, expirá-los, conflitos de gravação, entre outras ações.

"Uma linha é uma coleção de colunas anexadas ou associadas a uma chave; uma coleção de linhas semelhantes constitui uma família de colunas." (SADALAGE; FOWLER, 2013, p. 149). As famílias de colunas simples são denominadas família de colunas padrão. Novas colunas podem ser adicionadas a qualquer momento, em qualquer linha.

Temos uma supercoluna quando há a existência de um conjunto de colunas. Esta, possui um nome e um valor, sendo seu valor o mapa de colunas. Pode se criar uma família de supercolunas com o intuito de relacionar todos os dados agrupados. Essas famílias de colunas são colocadas em um *keyspace.* "Um keyspace é semelhante a um banco de dados em SGBDRs, em que todas as famílias de colunas relacionadas ao aplicativo ficam armazenadas." (SADALAGE; FOWLER, 2013, p. 151). Um *keyspace* pode ser criado a partir do seguinte comando:

create keyspace [nome]

Onde [nome] deve ser o nome que o cliente desejar utilizá-lo para que a família de colunas seja atribuída.

No Cassandra, assim que uma gravação é efetuada, ela é armazenada em um *commit log*, e após isso, aplicada em uma estrutura na memória denominada *memtable*. A operação de gravação somente é considerada bem-sucedida se forem gravados os dados no *commit log* e no *memtable*. Ainda, se um nodo ficar indisponível, os dados que deveriam ser armazenados nele serão armazenados temporariamente nos outros nodos e, assim que o nodo se reestabelecer e ficar disponível novamente, os dados serão retornados para que ele possa armazená-los. Essa técnica é denominada como *hinted handoff*.

No Cassandra, a atomicidade só é garantida na gravação da linha. Caso a inserção ou modificação venha a afetar várias colunas de acordo com a chave de linha, esta, será tratada como uma única gravação ou modificação, mas poderá ter sucesso ou não, ou seja, a atomicidade nesse caso não é garantida.

As consultas aos dados inseridos no Cassandra se dão dentro dos *keyspaces* criados. Primeiramente é necessário utilizar o comando *use [nome]*, onde [nome] é o nome que o cliente utilizou na criação do *keyspace*. Dessa forma, estamos dizendo ao SGBD que iremos fazer consultas apenas dentro deste *keyspace.* Digamos que, por exemplo, tenhamos uma linha chamada de Aluno. Queremos trazer todas as colunas que pertencem a um aluno cujo nome cadastrado seja Steve Braga. Dessa forma utilizaremos o comando GET, seguido do nome da linha e a especificação do que queremos, então temos:

GET Aluno[‘Steve Braga’];

Ainda, é possível somente trazer uma coluna, dentro do conjunto ou família de colunas.

GET Aluno[‘Steve Braga’][‘RA’];

Utilizando este comando, estamos apenas solicitando o RA vinculado ao aluno informado no comado. Para atualizar informações no Cassandra, é utilizado o comando SET. Tomaremos como exemplo ainda a linha Aluno, e atualizaremos o RA do aluno acima. Assim, temos:

SET Aluno[‘Steve Braga’][‘RA’]=’173020371’;

Para que possamos remover colunas, ou um conjunto inteiro delas, utilizamos o comando DEL. Dessa forma:

DEL Aluno[‘Steve Braga’][“RA’];

DEL Aluno[‘Steve Braga’];

O primeiro comando apenas remove o valor dentro da coluna RA. O segundo, remove todas as colunas que estiverem inseridas dentro de ‘Steve Braga’.

O Cassandra possui sua própria sublinguagem de consulta e suporta funções SQL, conhecida como CQL ou *Cassandra Query Language*. A utilização da criação de família de colunas é semelhante à criação de tabelas na linguagem SQL. Inserir e consultar dados também são utilizados de forma semelhante à SQL, contudo, o CQL não possui suporte à junções e suas atribuições de WHERE são muito restritas.

A escalabilidade do Cassandra e dos outros SGBD’s são feitas de modo horizontal. “Como nenhum nodo é mestre, quando adicionamos nodos ao cluster,

estamos melhorando sua capacidade de suportar mais gravações e leituras.” (SADALAGE; FOWLER, 2013, p. 158). Acontece que os nodos podem ainda serem adicionados em qualquer momento sem abdicar do *uptime* do serviço, ou seja, não é necessário desligar a aplicação para que mais nodos sejam adicionados.

O modelo de armazenamento em famílias de colunas também é recomendado no caso da aplicação ter a necessidade de um registro de *logs* pois podem armazenar quaisquer estruturas de dados e escalam as gravações para que se tenha uma linha de tempo de registros de eventos.

Sistemas de gerenciamento de conteúdo e plataformas de blogs também se aproveitam deste modelo visto que “[...] você pode armazenar entradas de blogs com tags, categorias, links e trackbacks em diferentes colunas.” (SADALAGE; FOWLER, 2013, p. 159). Este modelo também é uma boa opção quando se quer contar o número de visitantes de um *website,* por exemplo. Você pode incrementar o número do contador de visitantes na linha sempre que este visitar a página.

A necessidade de expirar programas, banners ou propagandas também podem ser gerenciadas pelo Cassandra. O suporte ao período denominado *Time to Live* (TTL) – tempo para viver, dado em segundos, dá ao Cassandra a possibilidade de seu usuário utilizá-lo para automaticamente deletar colunas assim que esse tempo esgotar-se.

O modelo de dados de famílias de colunas podem não ser a melhor opção quando a aplicação requer transações ACID, que consiste em Atomicidade, Consistência, Isolamento e Durabilidade. Atomicidade garante que os dados serão inteiramente gravados, ou devolvidos com erro, sem que uma parte deles sejam inseridas indevidamente no banco de dados. A consistência é um parâmetro que visa a consistência dos dados inseridos no banco. Por exemplo, não podemos deixar o banco de dados retirar do estoque de uma empresa 10 mil produtos se apenas há em estoque 5 mil produtos, o que geraria inconsistência no sistema. Isolamento significa que todas as operações serão efetuadas distintamente uma da outra e de forma que duas nunca possam ser efetuadas ao mesmo tempo. Durabilidade é a consideração dos dados para que não sejam modificados ou excluidos de forma inesperada, apenas em casos que o administrador ou aplicação realmente desejem ou necessitem da exclusão ou modificação desses dados.

4.6 O MODELO DE GRAFOS

Os SGBD's mais conhecidos para este modelo são o Neo4J, Infinite Graph, OrientDB e o FlockDB, sendo este último exclusivo dentro do modelo, pois possui a característica de que apenas um relacionamento pode ser definido para cada aresta.

"Bancos de dados de grafos permitem que você armazene entidades e também relacionamentos entre essas entidades." (SADALAGE; FOWLER, 2013, p. 161). Esse modelo é constituído por arestas que se relacionam entre si, também conhecidas como nodos. Esses nodos são direcionais e podem ser bidirecionais em alguns casos. Não existe limitação quanto ao número de relacionamentos que os nodos podem ter.

A consulta nesse modelo é denominada travessia, e a sua vantagem é que "[...] podemos alterar os requisitos de travessia sem ter de alterar os nodos ou arestas." (SADALAGE; FOWLER, 2013, p. 162).

O modelo de grafos não suporta a distribuição de nodos em servidores diferentes, exceto o Infinite Graph. Contudo, eles ainda possuem a ferramenta de fragmentação entre o nodo mestre e os nodos escravos. Enquanto os escravos estão sempre disponíveis apenas para a leitura, uma gravação no mestre é replicada imediatamente aos escravos, já uma gravação efetuada no escravo será replicada imediatamente ao mestre, mas não aos escravos. Somente o mestre, assim que receber todos os dados e gravá-los, efetuará a replicação para os nodos escravos.

O Neo4J é compatível com as transações ACID, e se faz necessário no código que esteja explícito que a transação foi bem-sucedida, caso contrário, o Neo4J negará os dados e não salvará nas arestas.

Para a utilização de consultas, os bancos de dados de grafos a gerenciam através de uma linguagem chamada de Gremlin. "Gremlin é uma linguagem específica de domínio para percorrer grafos; [...] O Neo4J também possui a linguagem de consulta Cypher para pesquisar o grafo." (SADALAGE; FOWLER, 2013, p. 167). Os nodos podem ser indexados para uma maior facilidade de travessia. Sendo assim, podemos consultar os dados dessa forma:

```
Node node = nodeIndex.get("nome","João").getSingle();
```

Estamos consultando os nodos que contenham o nome João como valor, a partir da indexação do mesmo. Podemos também consultar os relacionamentos que apontam para o João, por exemplo, possui, dessa forma:

```
incomingRelations = joao.getRelationships(Direction.INCOMING);
```

O modelo de grafos é útil quando é necessário à aplicação percorrer relacionamentos em níveis profundos, ou seja, todos os relacionamentos e consequentemente os relacionamentos dos relacionamentos.

Quanto à questão da escalabilidade desse modelo, ele não se aproveita da questão da divisão de dados entre servidores, uma vez que os nodos são relacionados diretamente entre si, faz mais sentido deixá-los em um só servidor. Para escalar o modelo de grafos, podemos adicionar RAM ao servidor, e este por si, armazenar os nodos trabalhados e relacionados inteiramente na memória. “Esta técnica somente é útil se o conjunto de dados no qual estivermos trabalhando couber em uma quantidade realista de RAM.” (SADALAGE; FOWLER, 2013, p. 172). Há também a possibilidade da adição de novos escravos, com acesso somente à leitura e todas as gravações replicadas ao mestre, que obtém a responsabilidade de replicar aos escravos.

Podemos fragmentar relacionamentos de acordo com o domínio dele. Nodos que se relacionam apenas com um grupo de nodos, e não com outros, podem ser fragmentados e disponibilizados em servidores diferentes, visto que a dificuldade essencial de fragmentação desse modelo, que é o relacionamento entre os dados, nesse caso não existe.

Redes sociais são as principais beneficiadas do modelo de grafos. “Qualquer domínio rico em links é apropriado para bancos de dados de grafos.” (SADALAGE; FOWLER, 2013, p. 173). Também, são apropriados para serviços baseados em localização, visto que, cada local e endereço pode ser um nodo, e, de forma inteligente, podem ser utilizados para dizer a menor distância entre o ponto de saída e o ponto de entrega. Mecanismos de recomendação também são recomendáveis a usar o modelo de grafos. Por exemplo, assim que um usuário compra algum produto, imediatamente os nodos armazenam essa informação, e informa aos amigos deste, recomendando a compra do produto. Esses dados armazenados crescem exponencialmente a medida que novas pessoas vão comprando novos produtos.

O modelo de grafos não é recomendado nos casos da necessidade de alterar propriedades em todos os nodos. Lidar com muitos dados também pode trazer problemas a este modelo, como a utilização de um grafo muito grande, uma vez que pode sobrecarregar o servidor em que o banco de dados esteja rodando.

4.7 ESCALABILIDADE NO MODELO NÃO-RELACIONAL

Como foi visto em todos os modelos inseridos no modelo não-relacional, existem inúmeras maneiras de se escalar o modelo não-relacional. Trabalhando com clusters, esse modelo torna-se fundamental à empresas que não possuem um hardware poderoso para lidar com grande quantidades de dados, e com isso, podem fazer do NoSQL a essência do funcionamento empresarial.

Podemos escalar de duas maneiras: indo pra cima ou para fora. “Ir para cima significa adquirir máquinas maiores, mais processadores, ter maior capacidade de armazenamento em disco e memória.” (SADALAGE; FOWLER, 2013, p. 32). Essa técnica é conhecida como escalar verticalmente, de forma que se adicione mais hardware aos componentes já existentes na infraestrutura computacional da empresa.

Ir para fora significa escalar horizontalmente o banco de dados, e assim, adicionar novos servidores fora da empresa para gerenciar os dados. Nesse sentido os modelos NoSQL suportam a fragmentação entre servidores, e elevar ainda mais o nível de escalabilidade dos sistemas, dado que, os servidores podem ser máquinas menores e mais acessíveis a pequenas empresas, mas que, conjuntamente, formam um grupo de clusters capazes de processar dados muito mais eficientemente do que apenas um servidor extremamente caro alocado na empresa. Ainda, o cluster oferece maior disponibilidade aos serviços, pois mesmo que uma das máquinas do cluster venha a ficar *offline*, as outras continuam em pleno funcionamento aguardando a outra voltar novamente a ficar disponível.

5 CONCLUSÃO

É notável o ganho que se obtém através da utilização de modelos de bancos de dados NoSQL, na tangente de escalabilidade dos sistemas. Muito embora os modelos relacionais ainda satisfaçam razoavelmente as necessidades das empresas, é preciso dar um voto de confiança ao modelo relacional. Caso haja a necessidade, é extremamente possível que se implante o modelo não-relacional apenas em uma parte do sistema, e na outra, o modelo relacional continue com o seu trabalho. Em certos casos, como em bancos de dados que exijam muitos relacionamentos, o modelo não-relacional torna-se inviável, ainda que possa ser implantado, mas não será conveniente, muito menos eficiente. Em outros casos, o modelo não-relacional pode substituir perfeitamente os modelos relacionais, e poderão impulsionar até outros setores da empresa, conforme o sistema facilite e agilize algumas partes da automação, os outros setores obtêm também respostas mais rápidas e podem realizar o serviço e as requisições mais rapidamente.

O fato é que esse modelo é extremamente recomendável em casos que não haja extrema necessidade de relacionamentos e a escalabilidade da infraestrutura atual da empresa esteja em desconforme com a necessidade capacitacional do banco de dados utilizado. Ainda, pode-se economizar tempo e dinheiro com a utilização de *clusters*, e assim distribuindo o processamento das requisições através do conjunto de computadores. O modelo surgiu pela necessidade da escalabilidade de sistemas, e fez o seu dever muito bem, trazendo diversas ferramentas aos seus SGBD’s para que tratem esse fundamento de forma eficiente, trazendo benefícios vastos às empresas que o implantam.

REFERÊNCIAS

DATE, C.J. **Introdução a Sistemas de bancos de dados.** Trad. Daniel Vieira. 8ª. ed. 9ª. reimp. Rio de Janeiro : Elsevier, 2003. 3-54p.

HERANNDES, Alex; NASCIMENTO, Jean. NoSQL – Você realmente sabe do que estamos falando?. **iMasters,** 28 maio 2010. Disponível em: <http://www.imasters.com.br/artigo/17043/banco-de-dados/nosql-voce-realmente-sabe-do-que-estamos-falando/>. Acesso em 29 abr. 2014, às 21h32min.

MACÁRIO, Carla Geovana do N.; BALDO, Stefano Monteiro. O Modelo Relacional. **Instituto de Computação da Unicamp**, Campinas/SP. Jan 2005. Disponível em: <http://www.ic.unicamp.br/~geovane/mo410-091/Ch03-RM-Resumo.pdf>. Acesso em: 28 out. 2014 às 14h44min.

SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL Essencial** : Um Guia Conciso para o Mundo Emergente da Persistência Poliglota . São Paulo/SP : Novatec, 2013. 5-175p.

SIMÕES, Gonçalo. Informação e Dados. **Aprender com as TICs**. Disponível em: <http://www.aprendercomastics.net/tic/materiaisapoio/Informacaoedados.pdf>. Acesso em: 26 set. 2014, às 15h34min.

O.K. Takai; I.C. Italiano; J.E. Ferreira . Introdução a banco de dados. **Instituto de Matemática e Estatística da Universidade de São Paulo,** São Paulo. Fevereiro 2005. Disponível em <http://www.ime.usp.br/~jef/apostila.pdf>. Acesso em 29 abr. 2014, às 22h44min.