FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Análise e Desenvolvimento de Sistemas

Henrique Mauricio de Albuquerque

NoSQL: CARACTERÍSTICAS E APLICAÇÕES

Americana, SP

2017

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Análise e Desenvolvimento de Sistemas

Henrique Mauricio de Albuquerque

NoSQL: CARACTERÍSTICAS E APLICAÇÕES

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Análise e Desenvolvimento de Sistemas, sob a orientação do Prof. Me. Wagner Siqueira Cavalcante.

Área de concentração: Desenvolvimento de Sistemas

Americana, SP

2017

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

A31n ALBUQUERQUE, Henrique Mauricio de

NoSQL: características e aplicações./ Henrique Mauricio de Albuquerque. – Americana: 2017.

58f.

Monografia (Curso de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Wagner Siqueira Cavalcante

1. NoSQL - banco de dados I. CAVALCANTE, Wagner Siqueira II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.3.07

Henrique Mauricio de Albuquerque

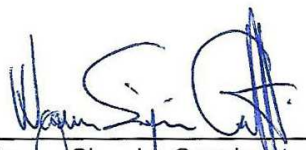
NoSQL: CARACTERÍSTICAS E APLICAÇÕES

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: Análise e Desenvolvimento de Sistemas.

Americana, 27 de junho de 2017

Banca Examinadora:



Wagner Siqueira Cavalcante (Presidente)
Mestre
CEETEPS/Faculdade de Tecnologia – FATEC/ Americana

Renato Kraide Soffner (Membro)
Doutor
CEETEPS/Faculdade de Tecnologia – FATEC/ Americana

Clerivaldo José Roccia (Membro)
Mestre
CEETEPS/Faculdade de Tecnologia – FATEC/ Americana

AGRADECIMENTOS

Agradeço aos meus exemplares pais que sempre estiveram comigo, me apoiando nas minhas escolhas e me incentivando com os estudos, aos amigos que me ajudaram de alguma forma, seja com dicas, fontes, métodos pessoais ou incentivo com palavras, ao meu orientador e um exímio mestre, Professor Wagner Siqueira Cavalcante que aceitou participar do meu projeto e contribuiu com sua sabedoria e ao meu supervisor, bem como ao José Tarcísio Cardoso que, além de um amigo, é também um excelente profissional que sempre me motiva.

RESUMO

O modelo de banco de dados relacional têm se mantido como solução unânime no armazenamento e manipulação de dados. O objetivo deste trabalho é apresentar e explicar novos conceitos e pensamentos de uma outra forma de armazenagem e processamento de dados, chamada *Not only SQL*, ou *NoSQL*, todavia deixando claro que uma nova tecnologia não substitui a outra, ou seja, existem necessidades distintas onde uma das soluções pode se sobressair. Para explicar os conceitos de agregados (termo apropriado em NoSQL), foi utilizada a linguagem de intercâmbio de dados *JSON*, que é bastante comum para a representação das coletâneas de dados. O presente texto também explica os tipos de banco de dados não relacionais e suas principais características, conceitos de modelagem e um detalhamento maior do Sistema Gerenciador de Banco de Dados (SGBD), *MongoDB.*

Palavras Chave: *NoSQL;Banco de dados;MongoDB*

ABSTRACT

The relational database model has remained the unanimous solution in data storage and manipulation. The objective of this work is to present and explain new concepts and thoughts of another form of storage and processing of data, called Not only SQL, or NoSQL, but making it clear that one new technology does not replace the other, that is, there are distinct needs where One of the solutions can stand out. To explain the concepts of aggregates (appropriate term in NoSQL), the JSON data interchange language was used, which is quite common for the representation of data collections. The present text also explains the non relational database types and their main characteristics, modeling concepts and a greater detail of the MongoDB DBMS.

Keywords: *NoSQL;DataBases;MongoDB.*

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

Durante um longo período o armazenamento de dados foi e ainda é realizado através de Sistemas Gerenciadores de Bancos de Dados (SGBD) Relacionais. Existem diferentes SGBDs tanto gratuitos quanto pagos que atendem praticamente toda a demanda de mercado. Quando se fala em armazenamento de dados, o modelo de banco de dados relacional está praticamente em todas as situações. Porém, devido ao surgimento de novas necessidades, grandes volumes de dados e a necessidade de se poder processar isso com alto desempenho, surgiu o SGBD não Relacional (SGBD NoSQL).

O NoSQL é totalmente distinto do modelo de banco de dados relacional e, assim, não se pode comparar duas plataformas que possuem propósitos diferentes, pois o NoSQL veio para suprir demandas que o Banco de Dados Relacional não consegue suprir de forma adequada. Esses novos modelos, entretanto, não são a melhor maneira para resolver todos os problemas referentes a armazenamento, e em muitos casos é preciso retornar à forma tradicional, utilizado por grandes empresas.

Para entender os conceitos de NoSQL e como ele funciona é necessário que haja um conhecimento prévio a respeito de Banco de Dados, pois este trabalho busca mostrar os conceitos, em quais situações é aconselhável e qual seria o ganho que pode-se obter aplicando-se o NoSQL.

1.1. JUSTIFICATIVA

O NoSQL é relativamente novo quando comparado com antigos conceitos da Tecnologia da Informação, porém recentemente se mostra em ascensão e vem sendo cada vez mais citado no mundo corporativo. Sua importância e contribuição para a área em si é justamente o aperfeiçoamento de busca e otimização constante de retornos, em um mundo onde se diz que tempo é dinheiro, é sempre bom ter algo otimizado e que se haja ganho de tempo, porém o intuito também é mostrar se esses ganhos são viáveis.

- Foco acadêmico: Para um tema que não está tão em voga é sempre mais difícil de se achar bons materiais, e é por este motivo que esta monografia poderá futuramente ser útil a diversos estudantes que desejam dar o primeiro passo e também incentivar professores e pesquisadores a criarem mais materiais a este respeito.
- Foco profissional: A importância desse estudo é justamente mostrar que existe outra alternativa referente a armazenamento de dados e também auxiliar empresas na tomada de decisão apresentando soluções que sejam efetivas no dia-a-dia.

1.2. MOTIVAÇÃO

Pequenas e grandes empresas podem possuir respectivamente pequenas e grandes bases de dados ou vice-versa. Ter uma base de dados repleta de informações não significa que o acesso à mesma poderá ser lento apenas pelo fato de ter um grande volume, ou seja, pode-se ter uma base bem estruturada com grandes volumes de dados e aos quais ter um acesso rápido, bem como um retorno com desempenho maior do que uma segunda base com menos volume e que não foi tão bem projetada.

Na situação atual, muitas pessoas que não tinham acesso ao computador, hoje possuem acesso aos smartphones e também aos computadores, aumentando exponencialmente a quantidade de pessoas que fazem o uso da tecnologia da informação. Devido a esse significativo aumento da quantidade de usuários, o número de requisições dos serviços alocados via *web* consequentemente aumentou.

1.3. OBJETIVOS

Para realizar este trabalho científico houve a necessidade de traçar objetivos rasos com foco nas mudanças de paradigmas em relação a banco de dados.

1.3.1. OBJETIVO GERAL

O objetivo dessa monografia é o de apresentar e explicar como funcionam os principais conceitos de NoSQL, e também simular cenários para que sejam

colocados em prática os conceitos apresentados e então fazer uma análise crítica das situações em que o NoSQL é recomendado.

1.3.2. OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho são:

- Apresentação da história e explicação das necessidades que tornaram o surgimento dos Bancos de Dados não Relacionais possível.
- Explicação dos principais conceitos como escalabilidade, chave de valor, grafos, orientação a colunas e documentos.
- Conhecer onde estes conceitos e em quais situações podem ser aplicados através de necessidades já existentes.
- Demonstrar como é feito através da simulação de cenários e da resolução na prática.
- Demonstrar algumas das vantagens e desvantagens da utilização desse modelo em relação ao Modelo Relacional.

1.4. METODOLOGIA

Revisão bibliográfica.

2. BANCO DE DADOS

Um banco de dados é uma coleção organizada de dados que segue regras, padrões e funções, os dados são inter-relacionados gerando um sentido e finalidade, porém o banco de dados não é gerido por si só e sim por sistemas gerenciadores de banco de dados (SGDB). Um SGBD é composto por diversos *softwares* com o objetivo de manipular e gerenciar os dados de forma organizada.

Desenvolvidos para gerir grandes quantidades de informação, fazendo com que seja reduzido o número de informações armazenado em meios físicos, como gaveteiros de arquivos, pastas entre outros. Por este motivo é que o sistema que fará a intermediação entre o banco de dados propriamente dito e o usuário deverá fornecer meios para manipular as informações armazenadas. Em boa parte dos casos possui informações confidenciais da empresa, por este motivo além dele ser projetado para armazenar as informações de forma organizada e otimizada, ele também precisará garantir a segurança das mesmas.

2.1. PRINCIPAIS APLICAÇÕES

Atualmente existem diversos tipos de situações que são mais bem solucionadas quando são utilizados. Geralmente essas situações envolverão cadastros de clientes, vendas, controle de pagamentos, agendamentos, relatórios entre outros.

A evolução da tecnologia da informação e o aumento do número de pessoas que a utilizam fizeram com que o crescimento da quantidade (e até da qualidade) de informação nos comércios existentes consequentemente aumentasse, sendo por este motivo que os Sistemas de Banco de Dados garantiram um imenso espaço no mercado.

Virtualmente, qualquer aplicação utiliza banco de dados, entre as quais algumas podem ser citadas:

- *e-commerce*: Além de efetuar os cadastros de clientes, são utilizados também cadastros de produtos, vendas, contabilização do estoque, cadastro de pedidos.

- *Hotéis*: *check-in* e *check-out,* quantidades e quais quartos estão disponíveis, reservas, serviços atribuídos à uma reserva, eventos.
- *Recursos Humanos*: Cadastro de funcionários e salários.
- *Faculdades e Escolas*: Registro acadêmico dos alunos, registro de professores, notas e cursos.
- *Bancos*: cadastro de conta corrente, empréstimos, clientes, extratos, cartões de créditos, faturas.
- Saúde: Hospitais, cadastro e histórico de pacientes, registro de medicamentos e quantidade adequada de consumo.

Pode-se perceber que esta lista é um pequeno fragmento das aplicações existentes, porém dá uma grande dimensão para pensar a respeito de quanto esses sistemas estão cada vez mais presentes nas simples tarefas do cotidiano.

No final da década de 1990 ocorreu a chamada “explosão da internet”, quando as empresas começaram a disponibilizar seus serviços via *web*. Os usuários muitas vezes não percebem que estão utilizando Banco de Dados e também nem possuem o contato direto com algum, mas os sistemas projetados para manipular as informações por sua vez dão a eles este privilégio, fazendo com que seja possível inserir, atualizar ou remover informações.

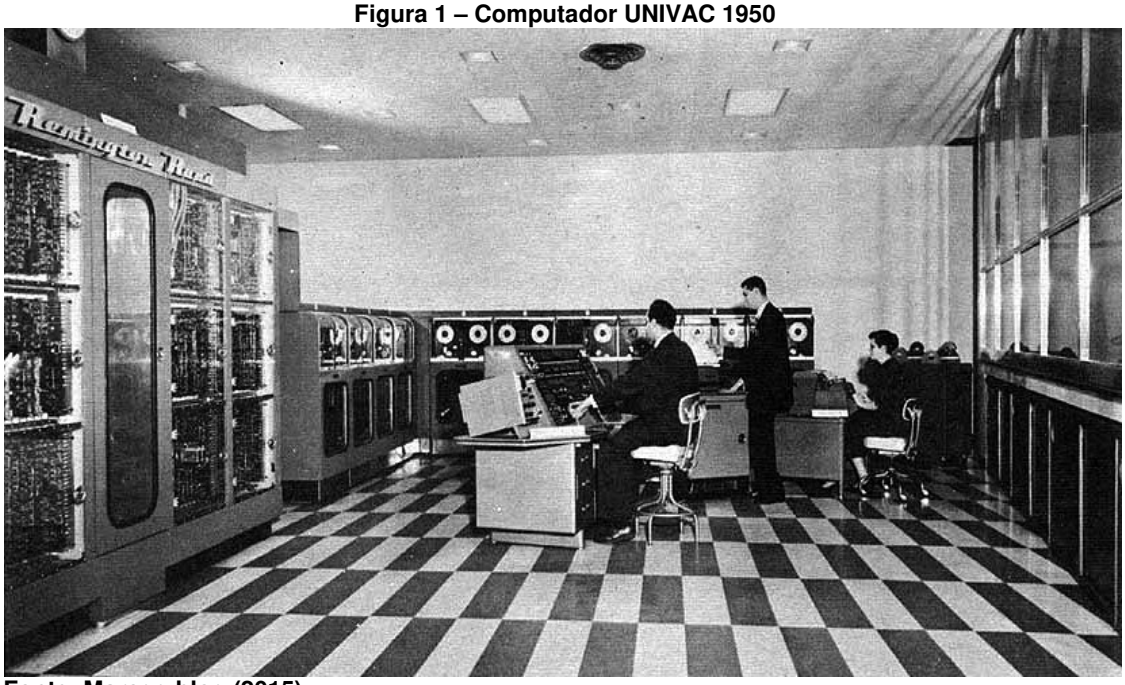
2.2. EVOLUÇÃO DOS SISTEMAS DE BANCO DE DADOS

Os meios de armazenamento e de processamento de dados passaram (e até hoje continuam passando) por diversas fases de evolução. O HD (*Hard Disk*) é um exemplo, pois hoje tem a capacidade de armazenar uma quantidade extremamente maior que os de antigamente. Com os sistemas de banco de dados não é diferente. Herman Hollerith inventou os cartões perfurados em papel, que foram usados muito antes de existir o computador, os quais passaram a ler as informações que eram perfuradas nos cartões.

O maior salto de evolução foi da década de 1950 até os anos 2000:

- **Final da década de 1950:** Grande parte do processamento de dados feito naquela época foi para o setor financeiro, quando os dados de salários, aumentos eram lidos a partir de fitas magnéticas e armazenado em outra fita. Os computadores também possuíam decks de cartão perfurado que após os dados serem lidos eram enviados para a impressão.

Figura 1 – Computador UNIVAC 1950



Fonte: Marcgg blog (2015)

- **Final da década de 1960:** Após o surgimento dos HDs houve um salto de melhoria imenso devido ao modo de armazenamento e processamento de dados. Devido possuir *clusters* e o modo como era feito o armazenamento, esses dados podiam ser acessados de qualquer parte do disco e com muito mais velocidade, sendo assim descartando o método de leitura sequencial. Com tal tecnologia em vigência podiam ser criados bancos de dados que utilizassem estruturas de dados como as listas e árvores.

- **Década de 1970 e 1980:** Nesse período houve a invenção do modelo relacional pelo matemático Edgar Frank Codd que a princípio aparentava ser excelente academicamente falando, porém não era utilizado no âmbito comercial devido a sua desvantagem de

desempenho perante aos bancos de dados de rede e hierárquicos. A empresa americana IBM sempre foi pioneira em diversos assuntos quando se trata de tecnologias, e com o banco de dados relacional não foi diferente, pois o setor de pesquisa desenvolveu o primeiro protótipo funcional chamado System R que resultou na criação do SQL/DS e logo depois surgiram os sistemas de banco de dados relacionais como o IBM DB2, Oracle, Ingres e DEC Rdb. Futuramente esses sistemas se equipararam com os antigos bancos de dados de rede e hierárquicos no quesito desempenho, porém a maior mudança ocorreu nas instruções que se tornaram mais simples e de mais alto nível, por que os mesmos executavam as tarefas de baixo nível automaticamente e os programadores não precisavam criar suas consultas de modo procedural utilizando mais tempo, ou seja, isso permitia que esses programadores pudessem trabalhar de uma maneira mais simples. O modelo relacional está presente na maioria das empresas até hoje.

- **Década de 1990:** O início da década foi marcado pelo desenvolvimento da linguagem SQL. No final devido à explosão da internet houve um aumento muito significativo nas aplicações de banco de dados e também nas transações, ou seja, os bancos de dados realmente começaram a ser utilizados de uma forma como nenhuma outra exigindo também 100% de disponibilidade dos serviços e também a integração com as interfaces *Web*.

2.3. MODELO ENTIDADE-RELACIONAMENTO (MER)

Formado por três conjuntos básicos, o modelo entidade-relacionamento (MER) foi criado para simplificar e facilitar o projeto de banco de dados. O MER auxilia no projeto de banco de dados, reduzindo a redundância (informações repetidas) e, consequentemente, possíveis inconsistências de dados. Desta forma, o modelo E-R se tornou a forma mais prática e de fácil abstração.

> "O modelo de dados entidade-relacionamento (E-R) foi desenvolvido para facilitar o projeto de banco de dados, permitindo especificação de um esquema de empresa que

> representa a estrutura lógica geral de um banco de dados." (SILBERSCHATZ, 2006)

2.3.1. ENTIDADES, ATRIBUTOS E RELACIONAMENTOS

Uma entidade é uma "instância" ou "objeto", distinto de outros objetos, tanto pertencentes ao mesmo conjunto de entidades, quanto pertencentes ao mesmo conjunto de entidades. Por exemplo, "Fulano de Tal" é uma entidade pertencente ao conjunto de entidades "Cliente", que possui características semelhantes a outros clientes, que os torna diferentes de "leite condensado", que é uma entidade pertencente ao conjunto de entidades "Produto", que, por sua vez, possui características próprias.

> "Uma entidade é um objeto que existe no mundo real e é distinguível dos outros objetos. Expressamos a distinção associando a cada entidade um conjunto de atributos que descreva o objeto." (SILBERSCHATZ, 2006)

Os atributos são características que descrevem as entidades. A tabela 1 possui 5 colunas, ou seja, são 5 características que estão atribuídas ao cliente do banco, definidas no projeto de banco de dados seguindo os requisitos que foram coletados. Neste caso, quatro atributos da tabela 1 poderiam se repetir, visto que pode existir um cliente com o mesmo nome e endereço, mas a coluna *id_cliente* é o número identificador de cada um que permite que os tornem únicos no mesmo conjunto de entidades. O atributo identificador estará presente em todas entidades, pois é através dele que as entidades se relacionam.

2.3.2. DIAGRAMA ENTIDADE / RELACIONAMENTO (DER)

O Diagrama Entidade-Relacionamento (DER) é uma representação gráfica dos principais aspectos do modelo E-R, permitindo um melhor entendimento para aquele que o observa.

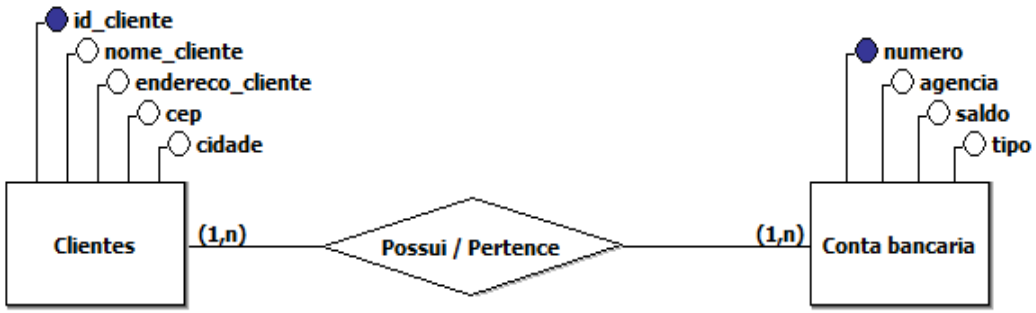
> "Os diagramas E-R são simples e claros – qualidades que podem ter motivado o amplo uso do modelo E-R." (SILBERSCHATZ, 2006)

Principais aspectos de um diagrama E-R:

- Retângulos: Representam as entidades.
- Losangos: Representam os relacionamentos.
- Círculos: Representam os atributos.
- Linhas: Responsáveis por fazer a ligação entre uma entidade e os relacionamentos.

A figura 2 demonstra o diagrama E-R que representa a entidade “Clientes” e “Conta bancária”, bem como seus atributos, ressaltando-se os identificadores “id_cliente”, do conjunto de entidades “Clientes”, e “número”, do conjunto de entidades “Conta bancária”. Neste caso, o DER da figura 2 representa a situação em que um cliente pode possuir uma ou mais contas bancárias (1,n), cada qual pertencente a um ou mais clientes (1,n):

Figura 2 – Diagrama Entidade-Relacionamento



Fonte: Próprio autor.

2.4. MODELO RELACIONAL

O modelo relacional consiste na utilização de tabelas e nos relacionamentos que estas realizam, representados por chaves que podem ser atributos identificadores dos registros de uma tabela, os quais são chamados de tuplas. As tabelas podem possuir diversas colunas e cada coluna representará um atributo da tupla em questão. As colunas possuem tipos de dados que podem ser texto, numérico inteiro, numérico com ponto futuante, entre outros, abordados na seção SQL deste capítulo.

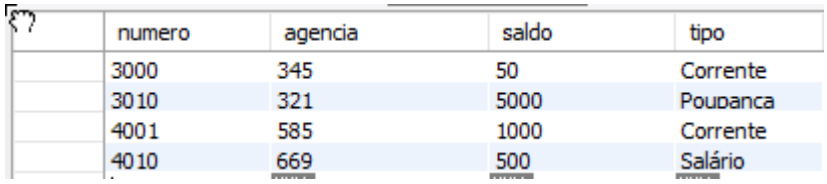
Criando-se o modelo Relacional, a partir do DER da figura 2, cada retângulo do DER (conjunto de entidades) converte-se uma tabela (figuras 3 e 4), cujas colunas serão compostas pelos atributos próprios (representados por círculos no DER), destacando-se o identificador, aqui intitulado de chave primária. Como este caso demonstra um relacionamento “muitos para muitos”, o losango (que representa tal relacionamento) também converte-se em uma tabela, cujas colunas são compostas pelos atributos identificadores de ambos os conjuntos de entidades relacionados, por isto intitulados de chaves estrangeiras. Pelo fato de ser uma nova tabela, esta precisa de uma chave primária que, além de permitir a identificação de uma tupla como diferente de todas as outras, impede a duplicação de sua existência, conforme observa-se na figura 5:

Figura 3 – Exemplo de clientes

id_cliente	nome_cliente	endereco_cliente	cep	cidade
1	Henrique	Rua Maria	13181	Sumaré
2	Jarbas	Rua João	10000	Campinas
3	Carlos	Rua Brasil	96054	Americana
4	Leena	Av. America	50000	Limeira

Fonte: Próprio autor.

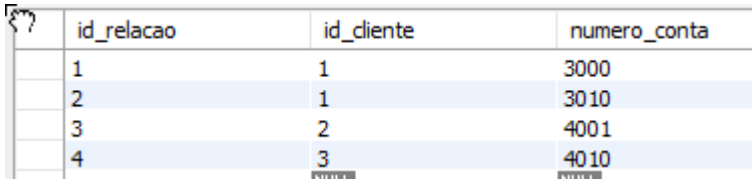
Figura 4 – Exemplo de contas bancárias



numero	agencia	saldo	tipo
3000	345	50	Corrente
3010	321	5000	Poupanca
4001	585	1000	Corrente
4010	669	500	Salário

Fonte: Próprio autor.

Figura 5 – Exemplo de relacionamento entre clientes e contas bancárias



id_relacao	id_cliente	numero_conta
1	1	3000
2	1	3010
3	2	4001
4	3	4010

Fonte: Próprio autor.

A figura 3 demonstra o registro de vários clientes e as colunas que representam os atributos que esses clientes possuem. Os atributos mudam de acordo com a necessidade e os requisitos de cada projeto de banco de dados.

A figura 4 possui os dados das contas bancárias que, como a tabela de clientes, esta guarda todas as informações a respeito do registro das contas bancárias.

A figura 5 representa uma relação entre cliente e conta. Neste caso pode-se observar que o cliente de número 1 possui 2 contas bancárias, ou seja, através dessa tabela é possível descrever à quem pertence as contas sem que haja redundância de informações. Por exemplo, o nome cliente aparecerá na tabela de contas para explicar a quem a conta pertence.

2.5. LINGUAGEM DE CONSULTA ESTRUTURADA (SQL)

A linguagem SQL (*Structured Query Language*) foi originalmente desenvolvida pela IBM, porém foi padronizada e efetivamente usada a partir da década de 90. Existem 5 porções da linguagem SQL, mas essa seção explicará melhor três delas:

- Linguagem de definição de dados (DDL): Essa parte é responsável pela definição propriamente dita, onde é possível criar, modificar e remover informações referentes ao esquema de banco de dados.

- Linguagem de manipulação de dados (DML): Responsável por realizar modificações, tanto inserções, quanto exclusões e modificações nos registros inseridos no banco de dados.

- Linguagem de consulta de dados (DQL): Após a criação do banco de dados, esta será a porção mais utilizada, através da qual são feitas todas as consultas.

> “Embora nos referimos a SQL como uma “linguagem de consulta”, ela pode fazer muito mais do que simplesmente consultar um banco de dados. Ela pode definir a estrutura dos dados, modificar dados no banco de dados e especificar restrições de segurança.” (SILBERSCHATZ, 2006)

Estas três porções da linguagem SQL serão usadas e são essenciais na construção de todos os bancos de dados relacionais.

2.5.1. DDL: COMANDOS BÁSICOS

Tomando-se como referência o DER que se encontra na figura 2, e consequentes tabelas do modelo relacional, descritos nas figuras 3, 4 e 5, esta seção mostrará exemplos de criação e modificação das tabelas utilizando os comandos DDL.

O comando *create table* é responsável por criar as tabelas que foram planejadas em etapas anteriores do projeto de banco de dados:

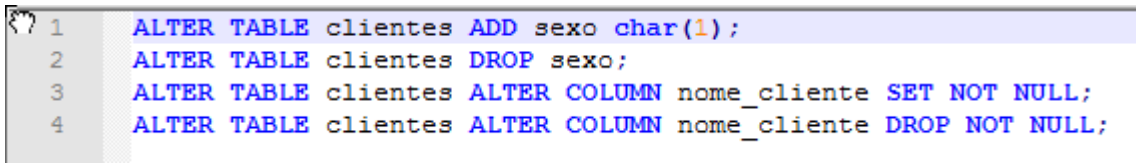
Figura 6 – Comando *Create Table*

```
Create table clientes
(  id_cliente integer,
   nome_cliente  varchar(50),
   endereco  varchar(50),
   cep  char(8),
   nome_cliente  varchar(50),
   constraint PK_C primary key (id_cliente)
);

Create table conta_bancaria
(  numero integer,
   agencia  integer,
   saldo numeric(12,2),
   tipo varchar(20),
   constraint PK_CB primary key (numero)
);

Create table cli_con
(  id_relacao integer,
   Id_cliente  integer,
   Numero integer,
   constraint PK_REL primary key (id_relacao),
   constraint FK_C foreign key (id_cliente) references clientes (id_cliente),
   constraint FK_N foreign key (numero) references conta_bancaria (numero)
);
```

Fonte: Próprio autor.

Para realizar alterações em uma tabela já criada, usa-se o comando *alter table*, que permite alterar a sua estrutura como, inclusão, exclusão de colunas ou de propriedades das mesmas.

Figura 7 – Comando *Alter table*



```
1  ALTER TABLE clientes ADD sexo char(1);
2  ALTER TABLE clientes DROP sexo;
3  ALTER TABLE clientes ALTER COLUMN nome_cliente SET NOT NULL;
4  ALTER TABLE clientes ALTER COLUMN nome_cliente DROP NOT NULL;
```

Fonte: Próprio autor.

Algumas propriedades do comando podem mudar de acordo com o SGBD utilizado. Os comandos utilizados como exemplo funcionam adequadamente no *PostgreSQL*.

3. NoSQL

O termo NoSQL (*Not Only SQL*) apareceu no final da década de 1990, em um banco de dados Open source chamado Strozzi NoSQL, projeto liderado por Carlo Strozzi. O nome NoSQL se dava pelo fato de o banco de dados não utilizar a linguagem SQL, que é padrão em bancos relacionais, em vez disso, tinha uma série de comandos rodados em *shell script* para Unix. Além do nome e do fato de não utilizar SQL como linguagem padrão, o banco de dados de Strozzi não muito tem em comum com os modelos de NoSQL que serão estudados ao longo deste trabalho científico.

Em 2009, mais especificamente em 11 de junho, Johan Oskarsson organizou uma reunião que difundiria o termo “NoSQL”.

> “Johan queria um nome para a reunião - algo que fosse um bom hashtag para o Twitter: curto, fácil de lembrar e sem muitos semelhantes no Google[..]selecionando “NoSQL”, de Eric Evans(Um desenvolvedor na Rackspace, sem conexão com Erick Evans do DDD - Domain Driven Design).” (PRAMOD, 2013).

Impulsionados por grandes projetos como Cassandra, *BigTable* e Dynamo, vários outros surgiram na comunidade NoSQL, nomes como MongoDB, Riak, Neo4j, Hbase, SimpleDB entre outros, ganharam espaço e valor em grandes empresas.

Apesar do que o nome pode sugerir, não é objetivo dos bancos de dados não relacionais acabar com o padrão SQL adotado pelos bancos relacionais, embora seja uma característica, o uso de linguagens próprias, recursos bem parecidos com

os utilizados no padrão SQL. Outra característica marcante é o fato de em sua maior parte, os projetos NoSQL são *Open Source*, embora existem empresas que mantenham projetos privados.

Este trabalho não tem como objetivo estudar a fundo todos os tipos e regras do banco de dados NoSQL. Ao invés disso, será apresentado um estudo sobre os diferentes tipos de banco de dados não relacionais como Chave-valor, Documentos, Família de colunas e Grafos, bem como os prós e contras de cada um e também algumas situações em que cada tipo de banco de dados possivelmente se enquadra melhor.

3.1. MODELO DE DADOS AGREGADOS

Por muitas décadas o modelo predominante tem sido o relacional, onde cada tabela possui linhas, que representam o dado completo sobre aquele objeto, e por colunas, que representam cada atributo que caracteriza cada objeto.

Quando o assunto é NoSQL, a mudança mais visível é seu afastamento do modelo relacional. Uma tupla é um conjunto de valores que de certa forma não é possível aninhar uma dentro da outra, ou seja, cada tupla pertencente a uma tabela é independente e pode-se formar uma lista com uma consulta que retorne os valores das tuplas, o que é uma particularidade do modelo relacional. A orientação agregada reconhece que o usuário deseja trabalhar com dados na forma de unidade e que é possível aninhar uma unidade de dados dentro de outra estrutura. É correto dizer que um agregado é um *container* de dados que não tem esquema pré-definido.

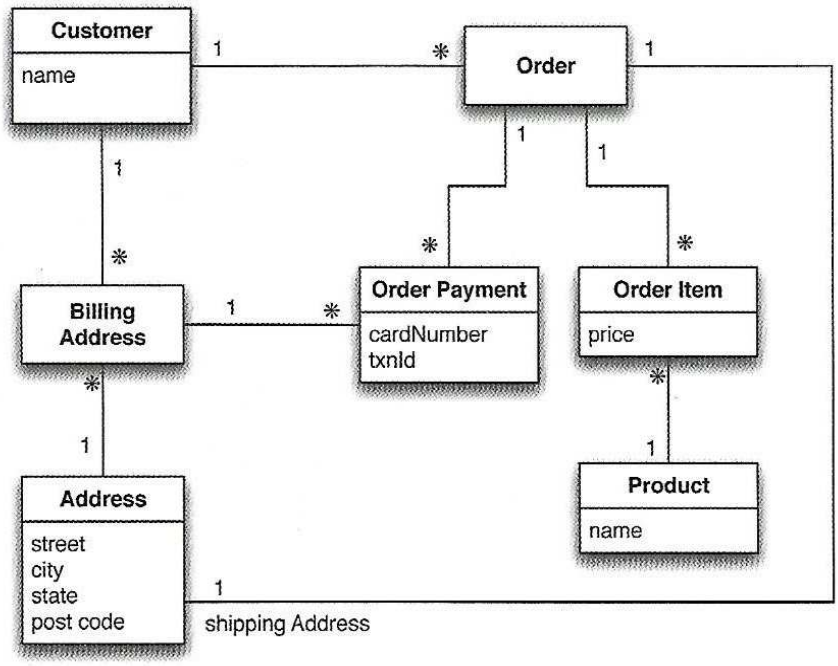
> “Lidar com agregados facilita muito a execução desses bancos de dados em um cluster, uma vez que o agregado constitui uma unidade natural para replicação e fragmentação.” (SADALAGE, 2013)

3.1.1. EXEMPLO DE RELAÇÕES E AGREGADOS

Neste trabalho será utilizado o modelo relacional para fins de comparação lógica e entendimento do NoSQL. Partindo do pressuposto que há uma necessidade de criar uma estrutura onde semelhante a de um comercio eletrônico com clientes,

endereços, pedidos, produtos e informações de pagamento (*Customer, Address, Products, Payment*), o que pode-se observar na figura 8:

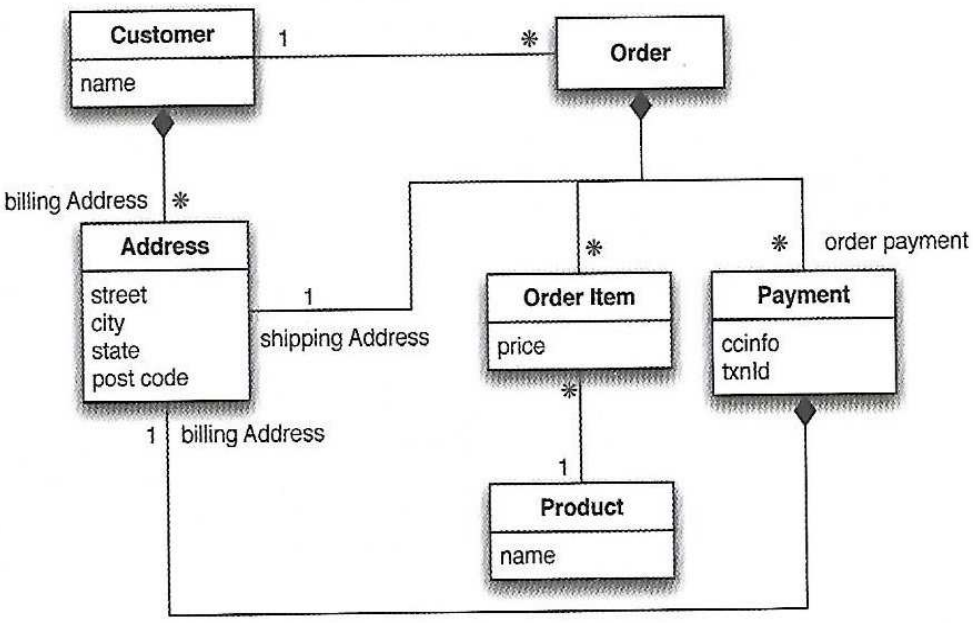
Figura 8 – Modelo de dados em um banco de dados relacional (notação UML)



Fonte: Sadalage e Fowler (2013)

A figura 9 demonstra um modelo de dados agregado.

Figura 9 – Um modelo de dados agregado.



Fonte: Sadalage e Fowler (2013)

A representação deste modelo de agregado é feita através da linguagem JSON, uma linguagem de intercâmbio de dados muito otimizada devido não possuir muitos cabeçalhos de informações como o XML, JSON é muito comum e muito utilizado na área de NoSQL, usada para representar a estrutura de um agregado. A figura 10 exibe um exemplo de representação em linguagem JSON do agregado apresentado na figura 9.

Figura 10 – Representação em linguagem JSON

```
// em clientes
{
    "id": 1,
    "name": "Henrique",
    "billingAddress": [{"city": "Tokyo"}]
}

// em pedidos
{
    "id": 99,
    "customerId": 1,
    "orderItems": [
    {
        "productId": 27,
        "price": 25.90,
        "productName": "DVD  Vol.1"
    }
        ],
        "shippingAddress":[{"city" : "Tokyo"}]
        "orderPayment":[
            {
                "ccinfo" : "'1000-1000-1000-1000",
                "txnId" : "abelif879rft",
                "billingAddress" : {"city": "Tokyo"}
            }
        ],
}
```

Fonte: Próprio autor.

Na estrutura de agregado, utiliza-se o losango preto para demonstrar que um cliente (*customer*) contém uma lista de endereços de cobrança (*billing address*); o pedido (order) contém uma lista de itens (*order item*) e o próprio pagamento (*payment*) contém um endereço de cobrança (*billing address*).

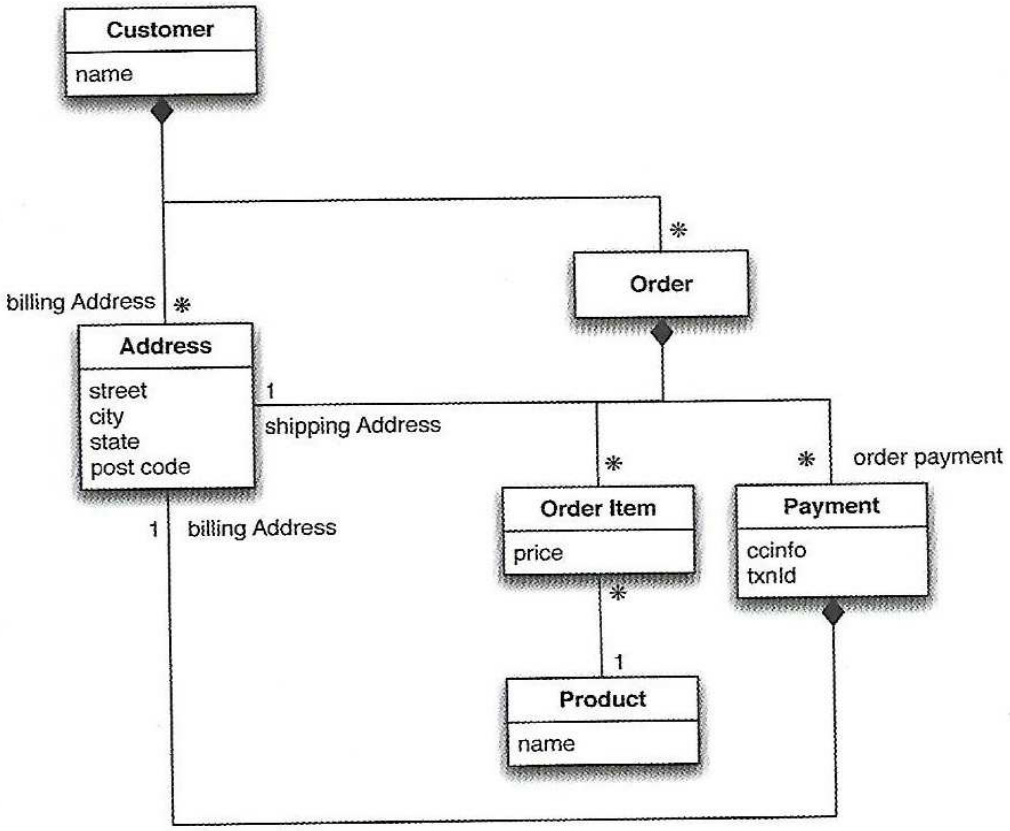
Nesse modelo, um único registro aparece três vezes, sendo que em uma base de dados relacional normalmente seriam utilizados números identificadores (IDs), porém esse registro é tratado como um valor e por sua vez é copiado, assegurando que, tanto os endereços de cobrança, quanto de envio e pagamento

não fossem alterados. Em um banco de dados relacional bastaria criar uma nova tupla. Com agregados, entretanto, deve-se copiar a estrutura de endereços pra o agregado.

Deve-se perceber neste conceito, não a forma como os agregados são definidos, mas como estes serão acessados.

Pode-se definir esse mesmo esquema de agregados de uma outra forma, como exibido na figura 11.

Figura 11 – Outra forma de definição para o agregado.



Fonte: Sadalage e Fowler (2013)

Na figura 12, pode-se observar a representação em JSON da figura 11 que possui um único agregado cliente (*customer*). O antigo agregado pedidos (*order*) exibido na figura 9 não está independente, e agora faz parte de cliente nesta representação.

Figura 12 – Representação em linguagem JSON

```
// em clientes
{
    "customer": {
        "id": 1,
        "name": "Henrique",
        "billingAddress": [{"city": "Tokyo"}],
        "orders": [
            {
                "id": 99,
                "customerId": 1,
                "orderItems": [
                    {
                        "productId": 27,
                        "price": 25.90,
                        "productName": "DVD Vol.1"
                    }
                ],
                "shippingAddress":[{"city":"Tokyo"}]
                "orderPayment":[
                    {
                        "ccinfo": "1000-1000-1000-1000",
                        "txnId": "abelif879rft",
                        "billingAddress": [{"city": "Tokyo"}]
                    }],
            }] // fecha orders
            } // fecha customer
} // fecha agregado
```

Fonte: Próprio autor.

Geralmente não há uma forma absoluta na modelagem para determinar a quantidade de agregados pois isso dependerá sempre da forma de como os dados serão manipulados ou o tamanho da aplicação. Pode-se optar por acessar todos os pedidos de um cliente de uma só vez com um único agregado ou acessar um único pedido por vez optando por ter vários agregados.

3.2. MODELO DE DADOS DE CHAVE-VALOR E DOCUMENTOS

Ambos os modelos lidam com agregados que são a base de uma estrutura não relacional, porém são completamente distintos. Para os bancos que utilizam o modelo chave-valor, estes descrevem o agregado como opaco. Descrevendo melhor, pode-se imaginar uma coletânea de bits sem significado, enquanto um banco de dados de documentos pode enxergar uma estrutura em um agregado.

Existem os prós e contra de cada modelo. Com um agregado opaco, ganha-se liberdade de população de dados.

> “A vantagem da opacidade é que podemos armazenar o que bem entendermos no agregado. O banco de dados pode impor algum limite de tamanho, mas de forma geral, temos liberdade completa.” (SADALAGE, 2013)

Um modelo de documentos exige uma definição de estrutura que é geralmente descrita em formato JSON e os tipos de dados que serão permitidos. Embora seja um pouco mais enraizado, perdendo-se a liberdade apresentada no parágrafo anterior, a vantagem é o ganho de mais flexibilidade no acesso, pelo fato de que a busca é realizada diretamente na estrutura determinada previamente.

> “Com um armazenamento de chave-valor, somente podemos acessar um agregado por sua chave. Com um banco de dados de documentos, podemos submeter consultas ao banco de dados baseadas nos campos do agregado.” (SADALAGE, 2013)

As formas de pesquisas entre esses dois tipos de banco de dados são bem similares. Ambos podem ter um ID e realizar a busca do agregado através de sua chave, enquanto o de documentos podem submeter uma consulta baseada na informação interna do agregado ou o agregado conter um ID e realizar uma busca no estilo chave-valor.

3.3. ARMAZENAMENTO DE FAMÍLIAS DE COLUNAS

O armazenamento de família de colunas é facilmente confundido com uma estrutura de tabela, mas esse seria um modo errado de se pensar.

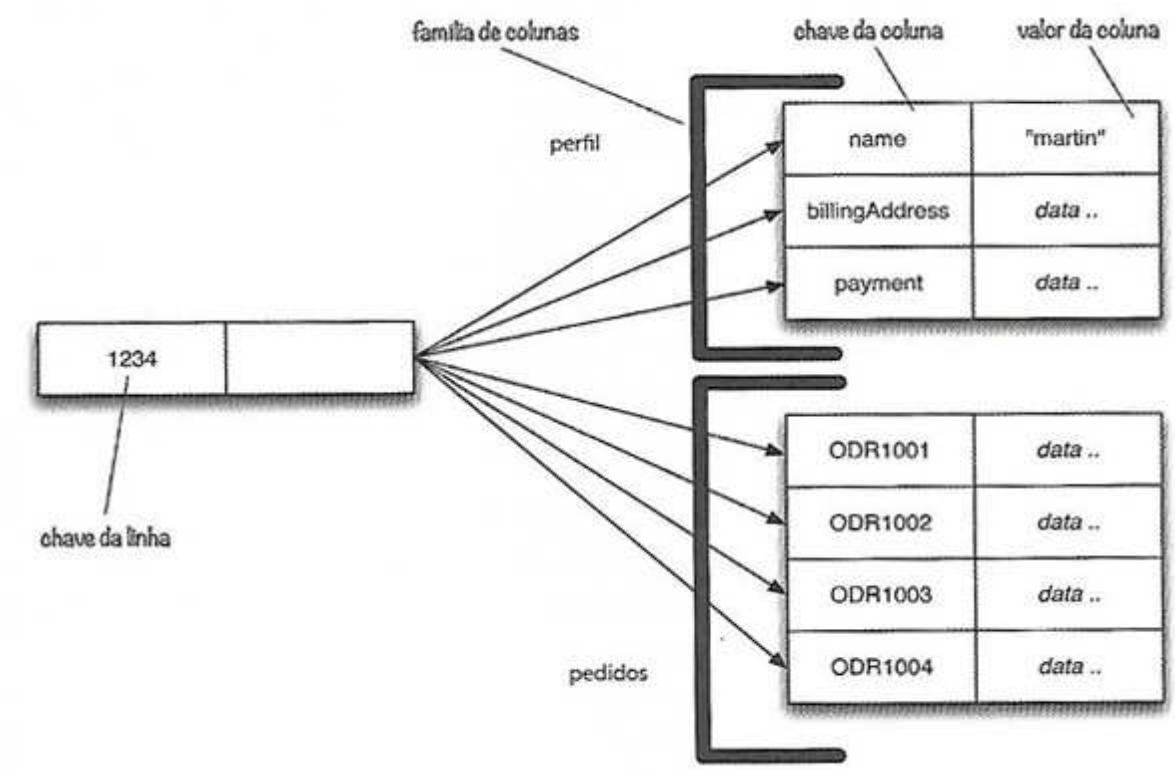
> “[...] não é bom pensar nessa estrutura como uma tabela, mas, sim, como um mapa em dois níveis.” (SADALAGE, 2013)

O *BigTable* do Google foi um dos primeiros bancos de dados NoSQL, e o que o tornou diferente foi justamente a forma como ele armazena os dados. Os bancos de dados já conhecidos utilizam a linha para armazenamento do registro, porém há situações em que não se terá muitas gravações, mas a todo momento necessitará ler colunas de muitas linhas de uma só vez, e é por este motivo que será necessário recorrer ao armazenamento de colunas.

Para exemplificar melhor, pode-se imaginar uma estrutura de chave-valor em que a chave é o identificador da linha, através do qual consegue-se selecionar o agregado em questão. As estruturas de colunas tem dois níveis: o primeiro nível é como a chave que identificará a estrutura de colunas, e o segundo corresponde às colunas propriamente ditas. É relevante dizer que é possível selecionar apenas uma coluna em particular e não a estrutura inteira.

Na figura 13 observa-se uma linha com um identificador que indexa toda a estrutura, porém é possível fazer uma solicitação de uma única coluna, executando-se o código “get(‘1234’, ’name’)”.

Figura 13 – Representação de uma estrutura de família de colunas



Fonte: Sadalage e Fowler (2013)

3.3.1. CASSANDRA E DEFINIÇÕES DE TIPOS DE LINHAS

Comparado ao *BigTable* (Google), o Cassandra (Facebook) possui uma visão um pouco distinta. No Cassandra uma linha possui uma família de colunas, porém

essa família de colunas pode conter supercolunas, ou seja, colunas que possuem mais colunas aninhadas.

> "Ainda pode ser confuso pensar em famílias de colunas como tabela. Você pode adicionar qualquer coluna a qualquer linha e as linhas podem ter chaves de coluna muito diferentes." (SADALAGE, 2013)

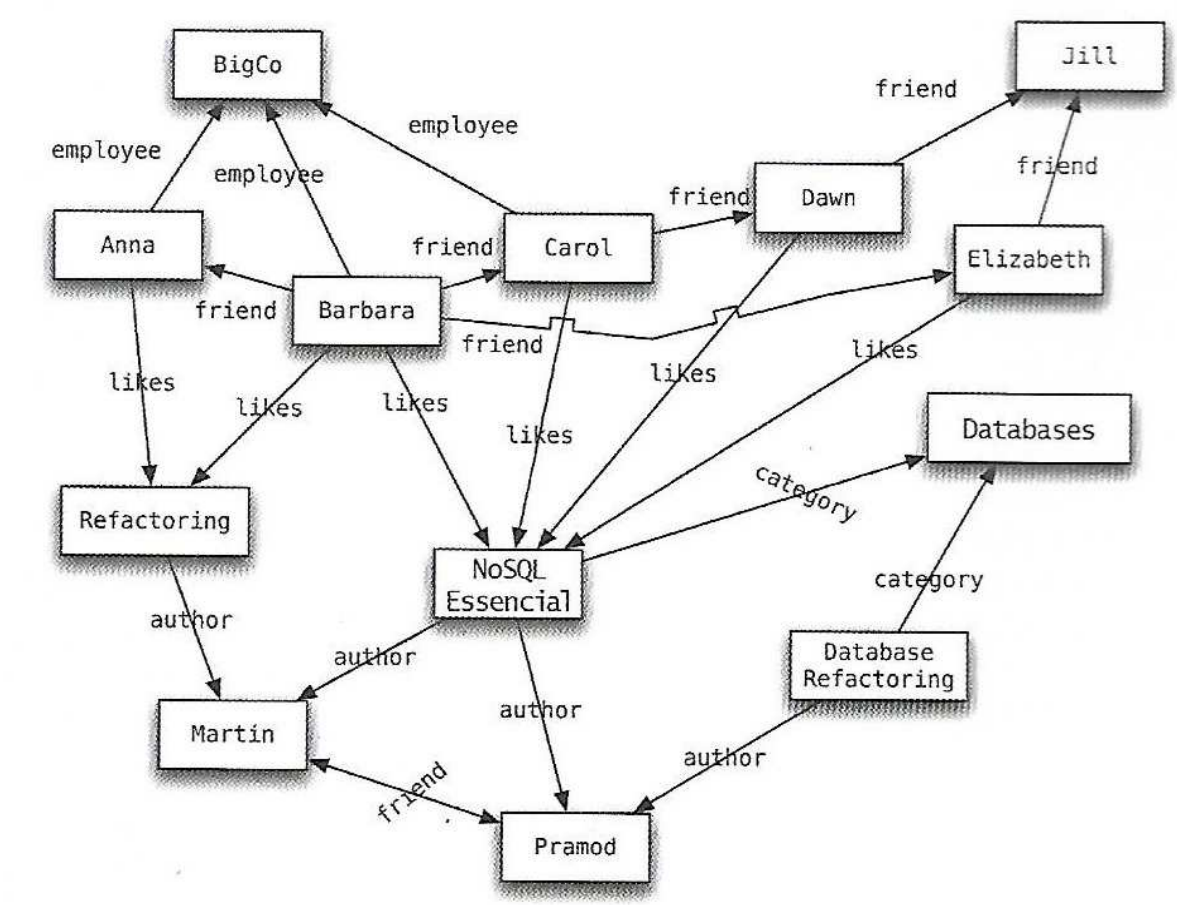
O Cassandra define como linhas estreitas as estruturas que possuem poucas colunas e essas mesmas colunas são utilizadas pelas diferentes linhas que ela possui. Para essa atribuição, a família de colunas define um registro; cada linha do registro é uma coluna e cada coluna é um campo. Linhas largas ao contrário das estreitas possuem muitas colunas, que por sua vez modela uma lista e cada coluna se torna um elemento dessa lista.

3.4. BANCO DE DADOS DE GRAFOS (*GRAPHDB*)

Até o momento, os modelos de dados apresentados possuem conexões simples. Bancos de dados de grafos elevam essa discussão para um outro patamar onde eles acabam não sendo tão habituais no contexto de NoSQL, pelo fato de que utilizam registros pequenos com uma infinidade de conexões complexas. Um grafo é uma estrutura de dados de nodos simples conectados por arestas, as quais indicam ação/relação entre os nodos.

A figura 14 representa uma estrutura de grafo bem simples com um nodo minúsculo (apenas o nome), mas a estrutura, mesmo sendo simples em termos de tamanho, pode se tornar complexa por meio da quantidade de tipos de arestas que ela possuir.

Figura 14 – Exemplo de estrutura de grafo



Fonte: Sadalage e Fowler (2013)

Com essa estrutura é possível analisar o lugar de Jill que possui uma amiga chamada Elizabeth que gosta do autor Pramod. É possível fazer a seguinte solicitação, como “Encontre um livro da categoria banco de dados que foi escrito por algum autor de quem meu amigo gosta”, logo pode-se encontrar o nodo NoSQL Essencial.

Os bancos de dados de grafos são especialistas nesse tipo de captura de informação, porém elevado a um patamar em que um diagrama convencional não seria legível. Esse tipo de diagrama é perfeitamente usável quando é necessário utilizar relacionamentos complexos, como os de uma rede social.

Ao analisar as arestas, surge uma importante diferença entre os relacionamentos de um banco de dados relacional e os de um grafo. Um banco de dados relacional utilizaria chaves estrangeiras para realizar as relações entre as tabelas, porém quando houver muitas ligações pode ser um pouco custoso em

termos de desempenho, o que seria ruim para modelos de dados altamente conectados.

> “Bancos de dados de grafos compensam naturalmente em situações em que o desempenho da consulta é mais importante que a velocidade de inserção.” (SADALAGE, 2013)

Como já observado na figura 14, é possível realizar uma nova consulta como “me informe tudo que Carol e Dawn gostam”. Os dados estão na estrutura, e é por este motivo que é preciso determinar um ponto “x” de partida, que neste caso seria através do atributo ID que os nodos possuem para serem indexados na estrutura. A sequência lógica da busca seria procurar as pessoas chamadas “Carol” e “Dawn” e, a partir deles, seguir no uso das arestas.

4. MODELOS DE DADOS: CARACTERÍSTICAS E DETALHES

O uso de agregados é maior característica apresentada até o momento, porém existem outros aspectos que são importantes quanto à modelagem de dados.

4.1. BANCOS DE DADOS SEM ESQUEMA

Bancos de dados NoSQL são chamados ao redor de mundo de *Schemaless*. O termo em inglês reflete uma grande discussão em relação aos bancos de dados convencionais.

> “Ao armazenar dados em um banco de dados relacional, primeiramente deve-se ter um esquema, ou seja, uma estrutura definida para o banco de dados, que diz quais tabelas existem, quais colunas existem e quais tipos de dados cada coluna pode armazenar.” (SADALAGE, 2013)

O esquema de um banco de dados é o projeto propriamente dito, no qual estão descritas as informações mais importantes sobre o projeto, porém tratando-se de NoSQL, armazenar tais dados se torna uma ação muito mais informal. Ainda falando dos tipos de modelos apresentados nas seções anteriores, um armazenamento de chave-valor permite que sejam armazenados quaisquer tipos de dados sob uma chave. Um banco de dados de documentos faz o mesmo, devido

não possuir restrições relacionadas a sua estrutura. Em famílias de colunas, é permitido armazenar quaisquer dados com base em uma coluna especifica. Na estrutura de grafos é permitido adicionar livremente dados, arestas, nodos e propriedades aos próprios nodos e arestas.

Um benefício desse conceito é a liberdade e flexibilidade de se definir a estrutura com o projeto em andamento ou até mesmo em funcionamento. Um esquema exige um conhecimento muito grande do que precisa ser armazenado, porém isso pode ser algo difícil. Pode-se imaginar uma situação em que os requisitos não foram coletados de forma eficiente, ou o próprio cliente pode mudar um requisito, ou adicionar algo no andamento do projeto, fazendo com que a estrutura planejada anteriormente sofra mudanças significativas que acabam atrasando o cronograma do projeto. Sem um esquema pré-definido, é possível armazenar o que desejar, o que permite alterar o armazenamento de dados à medida em que se conhece mais sobre o projeto ou adicionar novas informações que surgirem ao longo do mesmo.

Um ponto importante, é que o armazenamento sem esquema traz facilidade para lidar com os dados não uniformes (dados em que cada registro possui um conjunto de campos diferentes). Um esquema enraíza os parâmetros da tabela, forçando-a a conter somente as colunas que serão utilizadas, caso contrário haverá uma tabela esparsa na qual haverá colunas sem significado algum para certos registros, enquanto outros registros dessa mesma tabela conterão tais dados nessas mesmas colunas.

4.2. VISÕES MATERIALIZADAS (*MATERIALIZED VIEWS*)

As visões materializadas surgiram para suprir uma necessidade quando se trata de agregados. Em banco de dados relacionais é muito simples executar, lidar com situações em que é necessário saber quanto um determinado produto vendeu em um determinado período, por exemplo. Porém, com agregados isso se torna algo complexo, forçando o administrador a ler todos os pedidos para obter a resposta. Um intermédio seria criar um índice no produto, mas seria totalmente contra a estrutura de agregados.

Visões (*views*) são bem comuns em banco de dados relacionais, sua função é fornecer um acesso que permita enxergar os dados de uma maneira diferente daquela como estão armazenados. Visões podem ser consideradas como uma tabela relacional, porém é definida através de uma instrução SQL e computada sempre que acessada, ou seja, pode ser considerada uma forma de encapsulamento, onde junta-se parte das tabelas para tornar o acesso mais flexível. Algumas visões podem exigir muitos recursos para serem processadas, então para contornar essa situação foram inventadas as visões materializadas (*materialized views*), que são visões computadas antecipadamente e inseridas em cache no disco, sendo muito usadas nos casos em que existem dados que precisam ser lidos diversas vezes.

> "Visões materializadas podem ser utilizadas dentro do mesmo agregado." (SADALAGE, 2013)

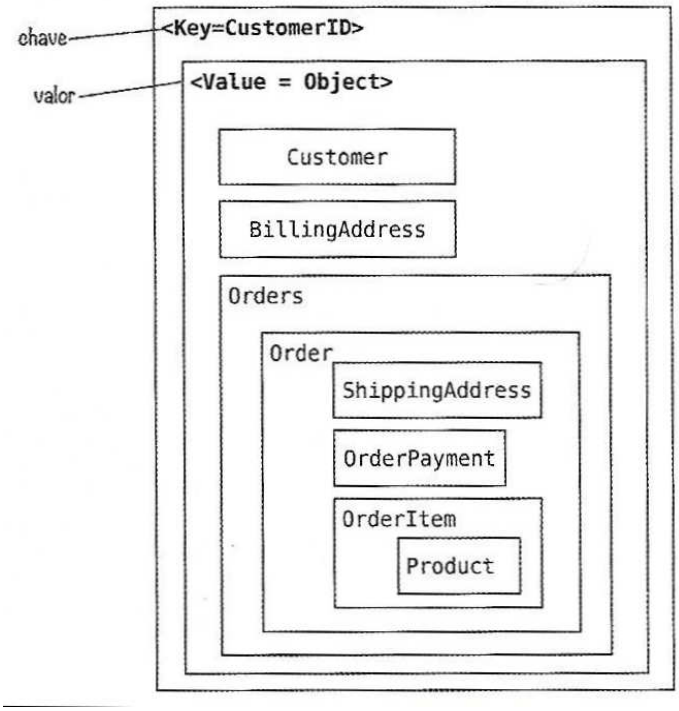
Um documento de pedido pode conter um resumo de pedido dentro de si, ou seja, o mesmo fornecerá informações resumidas sobre o pedido para que não seja necessário trazer todo o documento do pedido.

4.3. MODELAGEM

4.3.1. CHAVE-VALOR

Quando se fala em modelagem de agregados, é importante considerar como será o acesso aos mesmos. A figura 15 é um exemplo de chave-valor:

Figura 15 – Objetos de clientes e pedidos embutidos



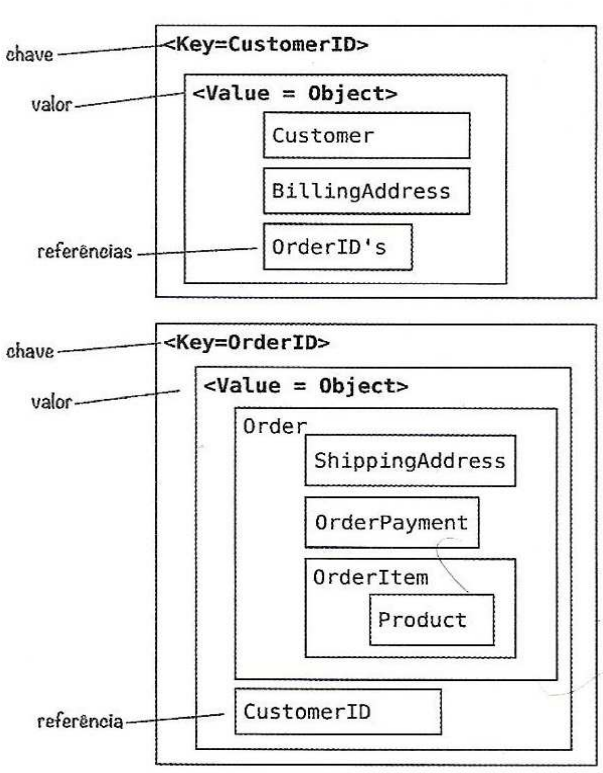
Fonte: Sadalage e Fowler (2013)

Para essa situação, a aplicação pode ler todas as informações do objeto e os dados que estão relacionados a ele utilizando a sua chave, porém, se houver uma solicitação de pedidos ou dos produtos que esses pedidos contêm, será necessário ler o objeto inteiro, visto que clientes e pedidos se encontram acoplados na mesma coletânea, ou seja no mesmo agregado. Para lidar com isso seria conveniente alterar para uma estrutura de documentos ou até mesmo manter o armazenamento de chave-valor, desde que os objetos clientes e pedidos estiverem separados, de modo que cada objeto terá um número identificador que referenciam entre si.

Trabalhando com a estrutura alterada entre clientes e pedidos, é possível encontrar todos os pedidos de um cliente. Um ponto positivo de trabalhar desta forma, é justamente a otimização da leitura desses dados, mas sempre que houver um novo pedido, essa referência deve ser enviada para o cliente no momento de inserção.

A figura 16 detalha como seria a estrutura de clientes e pedidos separados.

Figura 16 – Clientes e Pedidos separados



Fonte: Sadalage e Fowler (2013)

Para finalizar, a figura 17 define o exemplo da figura 16.

Figura 17 – Representação JSON da figura 16

```
# Objeto Customer (Cliente)
    {
        "customerID": 1,
        "customer": {
            "name": "Henrique",
            "billingAddress": [{"city": "Tokyo"}],
            "payment": [{"type": "debit", "ccinfo": "1000-1000-1000-1000"}],
            "orders": [{orderId:99}]
        }
    }

#Objeto Order (Pedido)
    {
        "customerId": 1,
        "orderId": 99,
        "order":
            {
                "orderDate": "Jan-06-2017",
                "orderItems": [{"productId": 27, "price": 29.90}],
                "orderPayment": [{
                "ccinfo": "1000-1000-1000-1000",
                "txnId": "abelif879rft"}],
                "shippingAddress": {"city": "Tokyo"}
            }
    }
```

Fonte: Próprio autor.

4.3.2. DOCUMENTOS

Tratando-se de armazenamento de documentos, visto que sua estrutura permite uma busca com base nos dados de cada documento, pode-se remover a referência de pedidos no objeto cliente. Essa mudança torna desnecessário atualizar o objeto cliente quando novos pedidos forem realizados.

Figura 18 – Modificação para estrutura de documentos

```
# Objeto Customer (Cliente)

    {
        "customerID": 1,
        "name": "Henrique",
        "billingAddress": [{"city": "Tokyo"}],
        "payment": [{
        "type": "debit",
        "ccinfo": "1000-1000-1000-1000"
        }]
    }
```

Fonte: Próprio autor.

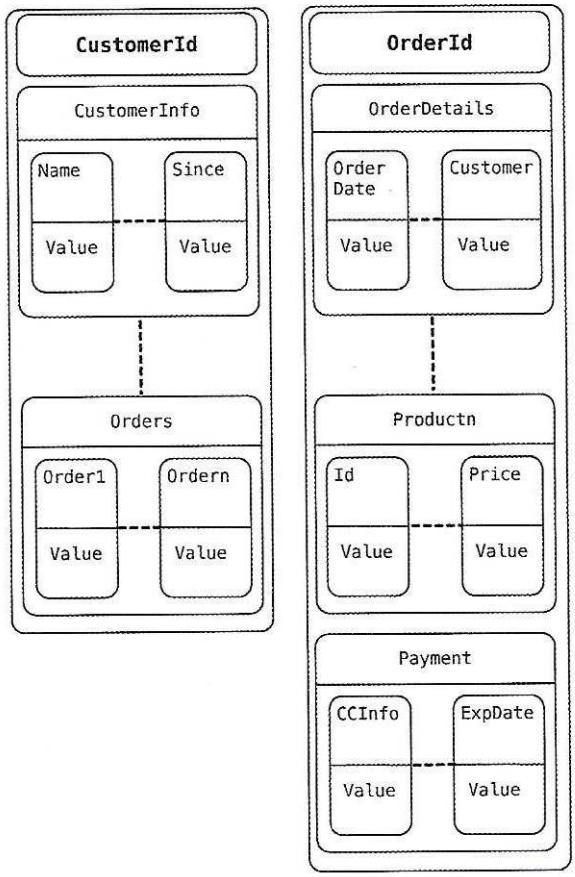
> "Uma vez que os armazenamentos de dados de documentos permitam que se faça a consulta pelos atributos de dentro do documento, pesquisas como "encontre todos os pedidos que incluam o produto X" são possíveis." (SADALAGE, 2013)

4.3.3. COLUNAS

Famílias de colunas têm o benefício de estarem ordenadas. É possível nomear colunas que sejam utilizadas com frequência, ou seja, esse benefício faz com que seja possível trazê-las primeiro. Para se obter desempenho, é preciso lembrar que se deve modelar pelas necessidades de consulta e não de gravação. Seguindo este princípio, tem-se a seguinte regra; facilitar consultas e desnormalizar os dados durante sua gravação.

Existem diferentes formas de modelar em todos os tipos de armazenamento, e com famílias de colunas não é diferente. Na figura 19, cliente e pedido (*Customer* e *Order*) serão armazenados em colunas diferentes. É importante observar que a coluna cliente tem as referências de todos os pedidos realizados.

Figura 19 – Modelagem de famílias de colunas



Fonte: Sadalage e Fowler (2013)

4.3.4. GRAFOS

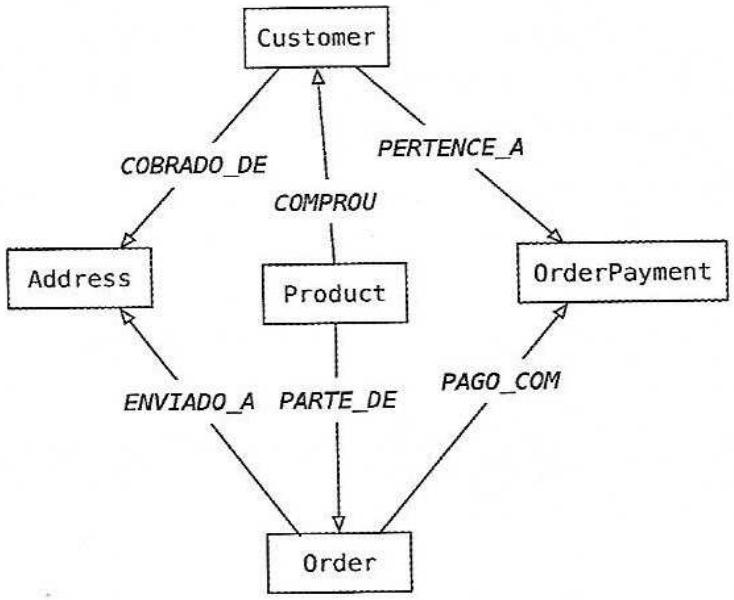
Para modelar os mesmos dados com grafo, pode-se adicionar os objetos como nodos e as ações e relacionamentos entre esses nodos por arestas.

As arestas ou relacionamentos são ações que podem ser lidas facilmente devido ao seu nome, isso ajuda na leitura fazendo com que seja fácil sua abstração. As ações “COMPROU”, “PARTE_DE, ENVIADO_A” da figura 20 reflete claramente a ideia de leitura. Um cliente “COMPROU” um produto que faz “PARTE_DE” um pedido que foi “ENVIADO_A” um endereço.

Para realizar buscas, basta encontrar o nodo necessário, como o endereço, por exemplo, para saber todos os pedidos que foram enviados para o endereço localizado que tem relacionamento de entrada com a ação ENVIADO_A. Um outro exemplo pode ser de produtos, localizando-se o produto em questão e realizando a

busca por intermédio da ação COMPROU que esse produto recebeu, tornando é possível encontrar os clientes que compraram determinado produto.

Figura 20 – Modelagem de gráfos



Fonte: Sadalage e Fowler (2013)

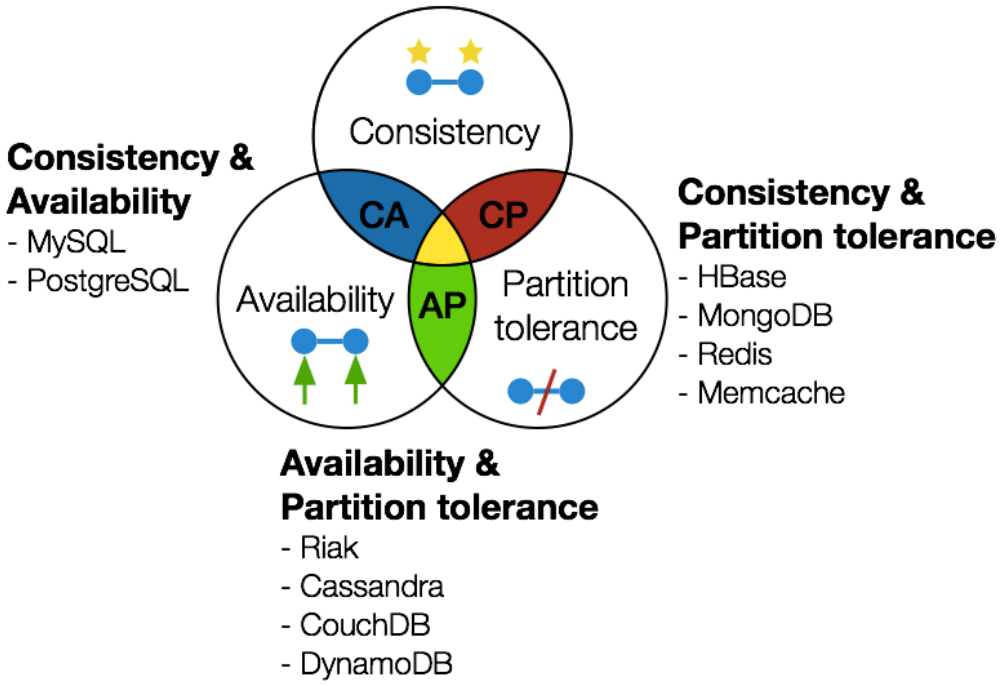
5. CONSISTÊNCIA DE DADOS

Os bancos de dados relacionais são fortemente conhecidos pela sua consistência, porém com o NoSQL a maior mudança está na forma de se pensar em relação à ela, por este motivo é possível de certa forma “configurar” o tipo de consistência necessária para cada situação.

5.1. TEOREMA CAP

Teorema CAP Proposto por Eric Brewer, em 2000, baseia-se em três propriedades: Consistência, Disponibilidade e Tolerância a partições, na ideia de que somente é possível obter duas delas.

Figura 21 – Teorema CAP



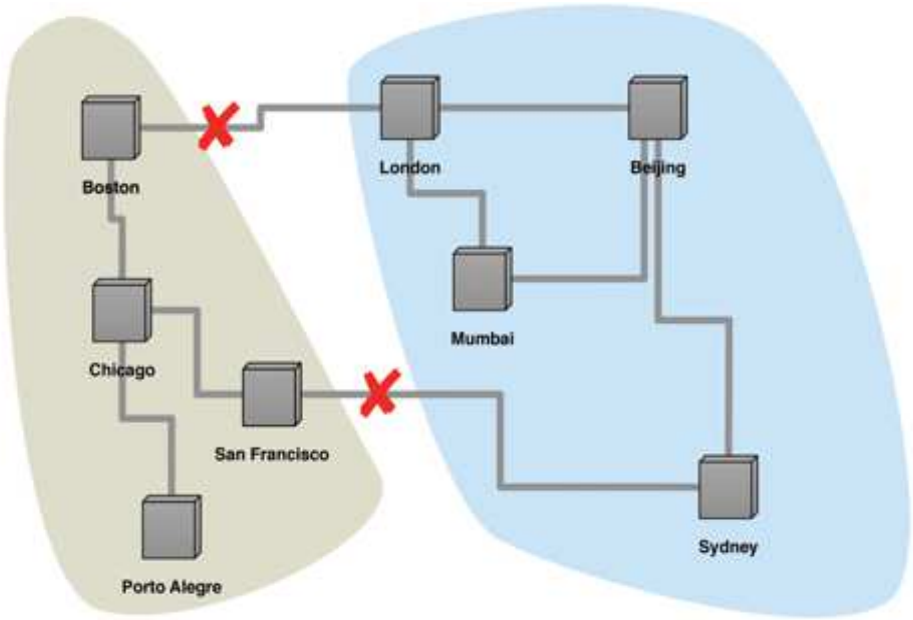
Fonte: Swapbytes (2016)

- **Consistência (*Consistency*)**: Bancos de dados relacionais são conhecidos pela sua consistência, que têm como objetivo manter a integridade das informações do banco de dados, dada a regra de negócio estipulada em sua criação. Para isso se utilizam do chamado ACID (Atomicidade, Consistência, Isolamento e Durabilidade - do inglês: *Atomicity, Consistency, Isolation, Durability*). Com esse conceito é possível trazer total segurança, devido garantir a atomicidade das transações, ou seja, garantir que todas as operações sejam executadas, ou que nenhuma operação seja executada em caso de falha, também conhecido como *Rollback*. Permite que o banco de dados passe de um estado consistente para outro estado consistente, garantido a manutenção de todas as regras de integridade. O BD garante o isolamento, ou seja, transações em paralelo nunca irão influenciar no resultado de outra. O BD também deve garantir durabilidade, uma vez que a transação termine em sucesso, o mesmo garante sua gravação.
- **Disponibilidade (*Availability*)**: No Teorema CAP, disponibilidade se resume a pelo menos um nó no Cluster, que deverá ser entendido

como uma máquina ligada à rede, ao qual se pode ler e/ou gravar dados.

- **Tolerância a partições (*Partition Tolerance*)**: Ser tolerante a partições significa que caso a rede à qual o nó está conectado se divida, cada parte seja capaz de conversar entre si.

Figura 22 – Tolerância a partições



Fonte: Sadalage e Fowler (2013)

> "No mundo NoSQL, é comum referirmo-nos ao teorema CAP como o motivo pelo qual pode-se precisar relaxar a consistência." (SADALAGE, 2013)

6. MONGODB

O MongoDB é um SGBD *OpenSource* que possui alto desempenho. Seu desenvolvimento se iniciou em 2007 e foi lançado publicamente em 2009. Ele foi escrito em linguagem C++, sem esquemas e formado por coletâneas de documentos JSON.

Figura 23 – Logotipo MongoDB

Fonte: MongoDB Documentation

6.1. EXECUÇÃO DE COMANDOS BÁSICOS

6.1.1. CRIAÇÃO E INSERÇÃO DE COLETÂNEAS DE DADOS

Operações de criação ou inserção (*Create, Insert*) adicionará novos documentos para uma coleção. Se não houver, qualquer operação de inserção irá criar uma nova coleção.

Figura 24 – Exemplo criação de coleção de dados

```
1
2   db.createCollection(inventario)
3       {
4           item: <string>,
5           quantidade: <number>,
6           tags: <array>,
7           tamanho: {
8           altura: <decimal>,
9           largura: <decimal>,
10          unidade: <string>,
11          }
12      }
13
```

Fonte: Próprio autor.

Pode-se criar uma coleção com uma instrução de inserção. No MongoDB é possível utilizar 3 métodos de inserção de documentos.

- *insertOne()* – Utilizado para inserir um único documento em uma coleção.
- *insertMany()* – Insere múltiplos documentos em uma coleção.
- *insert()* – Insere um único documento ou múltiplos.

Figura 25 – Exemplos de inserção

```
db.inventario.insertOne(
    {
        item: " mousepad ",
        quantidade: 100,
        tags: ["gel", "blue"],
        tamanho: { altura: 19, largura: 22.85, unidade: "cm" },
        status: "A"
    })

db.inventario.insertMany([
    {
        item: " mousepad modelo 1",
        quantidade: 150,
        tags: ["gel", "azul"],
        tamanho: { altura: 19, largura: 22.85, unidade: "cm" },
        status: "B"
    },
    {
    item: " mousepad modelo 2",
    quantidade: 100,
    tags: ["gel", "vermelho"],
    tamanho: { altura: 19, largura: 22.85, unidade: "cm" },
    status: "C"
    },
    {
    item: " mousepad modelo 3",
    quantidade: 25,
    tags: ["gel", "amarelo"],
    tamanho: { altura: 19, largura: 22.85, unidade: "cm" },
    status: "D"
    }]);
```

Fonte: Próprio Autor.

6.1.2. CONSULTAR DOCUMENTOS

Para selecionar dados de uma coleção, é preciso utilizar o método *find()*.

Caso não seja especificado nada dentro dos parênteses, o comando irá buscar tudo que houver dentro da coleção.

Figura 26 – Exemplo de seleção

```
// mongodb
db.inventario.find();

//SQL
SELECT * FROM Inventario
```

Fonte: Próprio Autor.

Existem muitos parâmetros, tanto no MongoDB, quanto na linguagem SQL. A figura 27 exibe um exemplo de seleção com parâmetro.

Figura 27 – Exemplos de seleção com parâmetro

```
// MONGO DB
db.inventario.find(
            {
                quantidade: 100
            });

// SQL
SELECT * FROM Inventario WHERE quantidade = 100
```

Fonte: Próprio Autor.

Pode-se utilizar as cláusulas “$lt” ou “$gt” que vem do inglês “*Less Than*” e “*Greater Than*”, que correspondem aos sinais de maior (>) e menor (<), conforme pode-se observar na figura 28.

Figura 28 – *Less than* e *greater than*

```
// MONGO DB
db.inventario.find(
            {
                quantidade: { $lt: 30 }
            });

db.inventario.find(
            {
                quantidade: { $gt: 30 }
            });

// SQL
SELECT * FROM Inventario WHERE quantidade < 100
SELECT * FROM Inventario WHERE quantidade > 100
```

Fonte: Próprio Autor.

Quando é necessário buscar algo com um parâmetro que possui mais de um valor, porém na mesma coluna, usa-se o comando SQL “*IN*” (figura 29).

Figura 29 – Exemplos de seleção com parâmetro "*IN*"

```
1  // MONGO DB
2  db.inventario.find(
3                      {
4                          status: { $in: [ "A", "B" ]}
5                      });
6
7  // SQL
8  SELECT * FROM Inventario WHERE status in ('A','B');
9
```

Fonte: Próprio Autor.

Se for necessário efetuar uma busca onde seja preciso filtrar por mais de uma coluna usa-se o conectivo lógico "*OR*", em português "OU" (figura 30).

O MongoDB também possui esta clausula, porém é escrita por "$or".

Figura 30 – Exemplo de seleção com parâmetro "*OR*"

```
41 // MONGO DB
42 db.inventario.find(
43                      {
44                          $or: [
45                              {
46                                  status: "A"
47                              },
48                              {
49                                  quantidade: { $lt: < 40 }
50                              }]
51                     });
52
53 // SQL
54 SELECT * FROM Inventario WHERE status = 'A' OR quantidade < 40;
```

Fonte: Próprio Autor.

Um tipo de dados que o MongoDB possui é o tipo *array,* ou seja, o mesmo é capaz de armazenar mais de um valor em um determinado campo. O *script* de inserção utilizado na seção anterior possui o campo *tags* que armazena 2 valores (figura 31), porém é possível armazenar mais, se necessário.

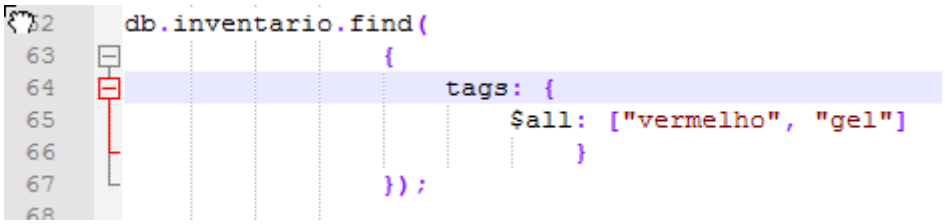
Figura 31 – Exemplo de seleção em um tipo *array*

```
56 db.inventario.find(
57                      {
58                          tags: ["gel", "vermelho"]
59                      });
60
```

Fonte: Próprio Autor.

A instrução apresentada na figura 32 busca exatamente o valor que o *array* possui e também em sua mesma ordem. Para filtrar independente desta ordem basta utilizar a cláusula “$all”.

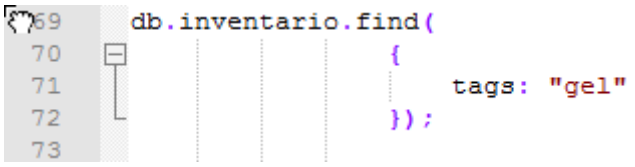
Figura 32 –Seleção em um tipo *array* com cláusula “*all*”



Fonte: Próprio Autor.

Alguns documentos podem conter mais de duas *tags,* por exemplo, “gel”, “vermelho”, “animal”, “cachorro”, sendo assim pode-se utilizar apenas o campo *tags* e na sequência não utilizar colchetes para determinar a cadeia de valores, isso fará com que o MongoDB busque em qualquer documento, independente da ordem que contenha a *string* solicitada.

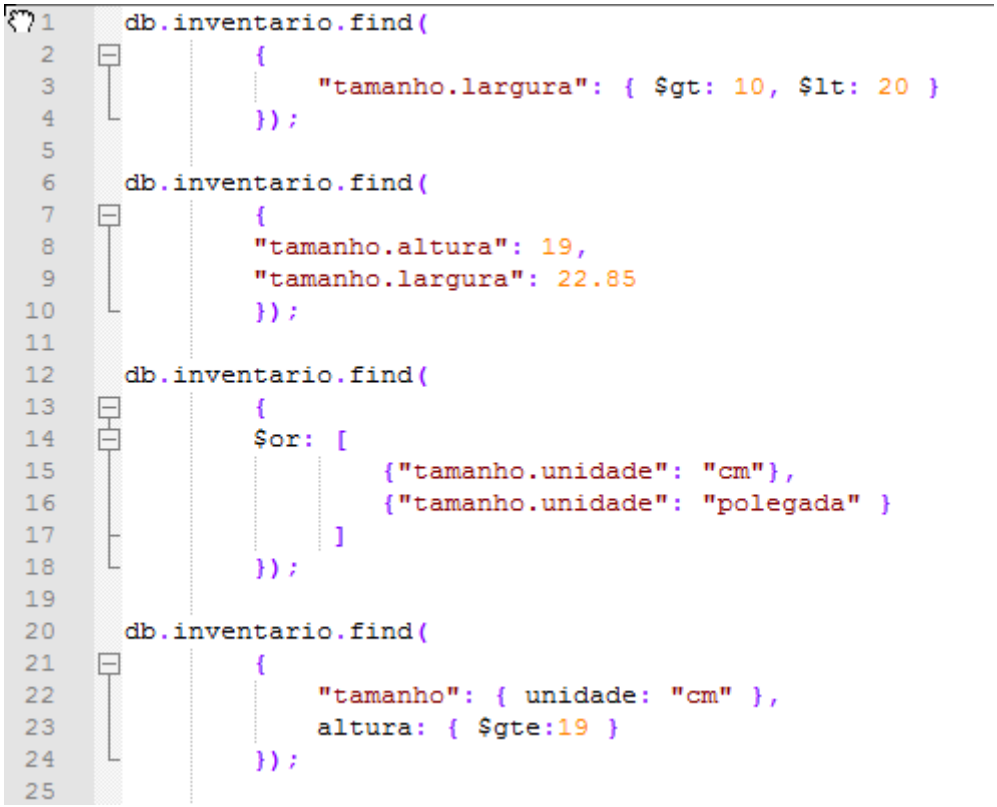
Figura 33 – Seleção em um tipo *array* sem parâmetro



Fonte: Próprio Autor.

Analisando o campo “Tamanho”, percebe-se que o mesmo é do tipo *array* e possui documentos embutidos, ou seja, um *array* que contém documentos e os mesmos documentos possuem valores.

Figura 34 – Seleção de documentos em um *array*



```
db.inventario.find(
        {
            "tamanho.largura": { $gt: 10, $lt: 20 }
        });

db.inventario.find(
        {
        "tamanho.altura": 19,
        "tamanho.largura": 22.85
        });

db.inventario.find(
        {
        $or: [
                {"tamanho.unidade": "cm"},
                {"tamanho.unidade": "polegada" }
            ]
        });

db.inventario.find(
        {
            "tamanho": { unidade: "cm" },
            altura: { $gte:19 }
        });
```

Fonte: Próprio Autor.

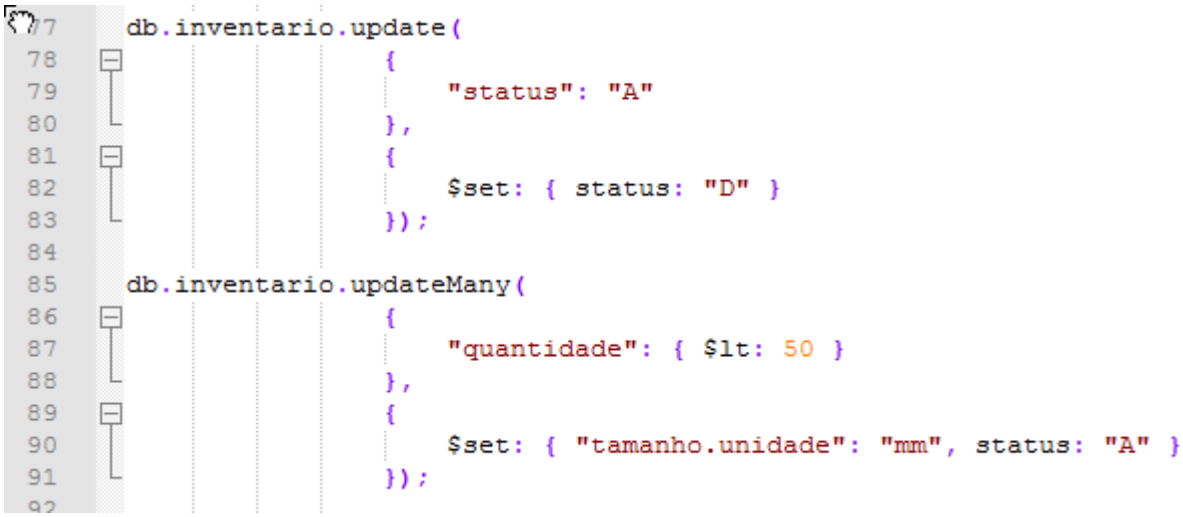
6.1.3. ATUALIZAR DOCUMENTOS

Existem 3 principais métodos de atualização.

- *update()* - Atualiza ou substitui um único documento que corresponde a um filtro especificado.
- *updateOne()* - Atualiza no máximo um único documento que corresponde a um filtro especificado, mesmo que vários documentos possam corresponder ao filtro especificado.
- *updateMany()* – Atualiza todos os documentos com um filtro especificado.

O primeiro parâmetro será o filtro e o segundo será o valor que o documento terá após a execução do comando.

Figura 35 – Exemplos de *update*



```
77  db.inventario.update(
78                  {
79                      "status": "A"
80                  },
81                  {
82                      $set: { status: "D" }
83                  });
84
85  db.inventario.updateMany(
86                  {
87                      "quantidade": { $lt: 50 }
88                  },
89                  {
90                      $set: { "tamanho.unidade": "mm", status: "A" }
91                  });
```

Fonte: Próprio Autor.

6.1.4. REMOVER DOCUMENTOS

Os métodos de remoção são simples de usar, são:

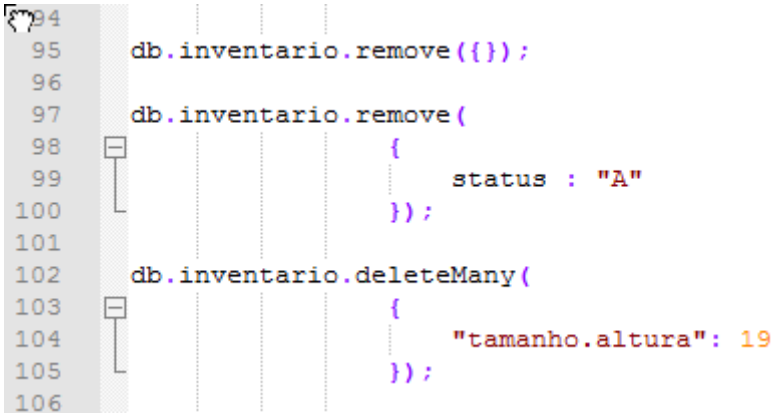
- *remove()* - Remove um único documento ou todos os documentos que correspondam a um filtro especificado.
- *deleteOne()* - Remove no máximo um único documento que corresponda a um filtro especificado, mesmo que vários documentos possam corresponder ao filtro especificado.
- *deleteMany()* - Remove todos os documentos que correspondam a um filtro especificado.

Para remover todos os documentos da coleção inventário criada na seção 6.1, basta utilizar o método *remove()*, sem especificar parâmetros.

Para remover todos os documentos de uma coleção pode ser mais eficiente utilizar o método *drop()*, que excluirá a coleção, incluído *indexes*.

A lógica de remoção é similar à de pesquisa, a qual utiliza um critério dentro do método para especificar onde a ação deverá ser feita. A figura 36 exibe exemplos de remoção.

Figura 36 – Exemplos de exclusão de dados



Fonte: Próprio Autor.

7. ESTUDO DE CASO

A empresa Globo.com, atuante no ramo de TV e entretenimento, possui diversas aplicações para comunicação, informação e interação para grande parte da população brasileira.

Uma de suas aplicações é o *Fantasy Game* Cartola FC, um jogo cujo tema é futebol que, por sua vez, é tratado por muitos como paixão nacional, envolvendo milhões de usuários. Os usuários possuem times fictícios e participam de ligas competitivas com resultados baseados no Campeonato Brasileiro (Brasileirão). Os resultados são obtidos através de escalações feitas pelos usuários, ou seja, se os jogadores escalados forem bem na rodada, o usuário faz mais pontos e, caso não forem, perdem pontos.

Rapidamente o Cartola FC se tornou o maior jogo do gênero no Brasil, e com isso, a Globo.com sentiu a necessidade de melhorar a interação entre usuários através de uma aplicação de *posts* (publicações) e *feeds* (mural de notícias).

7.1. PROBLEMA

O maior desafio para a empresa era criar uma aplicação de *feed* de alto desempenho e disponibilidade sem que houvesse atrasos nas respostas. A empresa utiliza o banco de dados relacional MySQL, que supre praticamente todas as demandas de suas aplicações. O MySQL sempre foi um Sistema Gerenciador de Banco de Dados robusto com grande capacidade de armazenamento, inserção e leitura de dados, porém para essa aplicação em especifico, o mesmo não mostrou ser a melhor opção.

O objetivo da empresa era fazer com que o usuário permanecesse na aplicação o maior tempo possível. O Cartola FC possui mais de 2 milhões de jogadores ativos e aproximadamente 90 milhões de *pageviews* (visualizações às páginas). Sendo assim, foi decidido que a maior premissa seria a velocidade da aplicação.

Os testes feitos com o MySQL, embora suprindo a necessidade de executar a aplicação, não foram satisfatórios em termos de velocidade de resposta. As consultas tiveram que ser refeitas, passando por um processo de *tuning* (otimização) e, mesmo assim, não alcançaram os resultados esperados.

Para trabalhar com esse volume de dados, a única opção que restou para o especialista de banco de dados, Franklin Amorim, foi utilizar um banco de dados não relacional, mas outros problemas surgiram com essa decisão: escolher um SGBD apropriado e gerenciar toda a equipe para migrar para uma outra tecnologia.

7.2. SOLUÇÃO ADOTADA

Ao se reunirem com o gerente de projetos foram definidas as premissas, além da velocidade já mencionada anteriormente:

- Robustez;
- Escalabilidade;
- Perda de mensagens não é um ponto crítico;
- Indisponibilidade inicial do serviço é tolerável;

Para atender os requisitos, optou-se por utilizar o MongoDB, um SGBD de documentos que se mostrou com as seguintes vantagens:

- Velocidade (mais rápido que o MySQL);
- Acesso aos dados (Sem escrita de consultas);
- Sem esquema;
- Sem necessidade de abstração de tabelas por parte dos desenvolvedores;

Com essa opção, foi possível criar uma aplicação com os mesmos padrões de *facebook* e *twitter*, com atualizações e respostas instantâneas para os usuários (figura 37), ou seja, a requisição era enviada e instantaneamente mostrada no mural de notícias de outros jogadores do mesmo grupo.

Figura 37 – Estrutura utilizada para o Cartola F.C.

```
{
    Data: "Tue Jun 15 2010 9:00:00 GMT-0400",
    Mensagem: "Minha mensagem",
    Liga: {
        id: 1930
        nome: "Liga 1930",
        logo: "Liga-1930"
        }
    Comentarios: [
        {
            Id: 251902,
            data: "Tue Jun 16 2010 11:00:00 GMT-0400",
            mensagem: "Legal!",
            time: { . . . }

        },
        {
            Id: 251903,
            data: "Tue Jun 16 2010 14:00:00 GMT-0400",
            mensagem: "Interessante!",
            time: { . . . }

        }
        ]
}
```

Fonte: Próprio autor.

Os resultados obtidos foram:

- Banco de dados sempre ativo e funcionando todos dias;
- Nenhum incidente referente ao desempenho desde sua implantação;
- 1 milhão de mensagens publicadas por mês.

O resultado de 1 milhão de mensagens foi abaixo do esperado, mas, através deste projeto foi possível diagnosticar outras aplicações que poderiam ser otimizadas com o MongoDB, como o site receitas.com e o catálogo de vídeos da empresa.

8. CONSIDERAÇÕES FINAIS

As pesquisas realizadas na área de banco de dados, mostram as mais diversas necessidades e a importância de uma escolha correta em relação ao armazenamento e processamento de dados.

Existem sistemas simples, como um cadastro de clientes, controle de estoque, um catálogo, e também sistemas mais complexos que são atendidos perfeitamente pelos bancos de dados relacionais, já estão no mercado há bastante tempo, os quais fazem um excelente trabalho.

Quando o assunto é banco de dados em larga escala como os que são utilizados por empresas como Google, Twitter, Facebook e Amazon que atendem uma quantidade de usuários a nível mundial, o NoSQL se encaixa perfeitamente para este tipo de demanda. Nos dias atuais existem cada vez mais dispositivos conectados à rede mandando requisições, recebendo respostas, com a necessidade de um rápido processamento, porém velocidade não é o fator primordial e sim a forma de armazenamento, qualidade e consistência de dados. Uma tecnologia relativamente nova que se mostra sólida, com melhorias constantes.

NoSQL pode ser a resposta para a maioria das dificuldades ou problemas em relação a armazenamento de grandes volumes de dados, e deve ser considerado como opção especialmente quando a aplicação irá trabalhar com estruturas complexas, ou quando a exigência por velocidade de exibição for maior que a de inserção.

9. REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, Adriano. Trabalhando com Relacionamentos: bancos de dados baseados em grafos e o Neo4j. Disponível em: <http://blog.caelum.com.br/trabalhando-com-relacionamentos-bancos-de-dados-baseados-em-grafos-e-o-neo4j/>. Acesso em: 10 fev. 2017.

ANDRÉ SANTANCHÈ. Bancos de Dados de Grafos (parte 1). Disponível em: <https://www.youtube.com/watch?v=zAQs3uhE1E8>. Acesso em: 12 fev. 2017.

ANDRÉ SANTANCHÈ. Bancos de Dados de Grafos (parte 2). Disponível em: <https://www.youtube.com/watch?v=zAQs3uhE1E8>. Acesso em: 13 fev. 2017.

DEVMEDIA. Introdução ao banco de dados NoSQL Cassandra. Disponível em: <http://www.devmedia.com.br/introducao-ao-banco-de-dados-nosql-cassandra/30533>. Acesso em: 01 fev. 2017.

DEVMEDIA. Conceito NoSQL: Cassandra em java. Disponível em: <http://www.devmedia.com.br/conceito-nosql-cassandra-em-java/22863/>. Acesso em: 02 fev. 2017.

IBM DEVELOPER WORKS. Considerações sobre o Banco de Dados Apache Cassandra. Disponível em: <https://www.ibm.com/developerworks/br/library/os-apache-cassandra/>. Acesso em: 02 fev. 2017.

MARCUS COSTA. Aula MongoDb – Conceitos básicos. Disponível em: <https://www.youtube.com/watch?v=mlUZJzFJNbM>. Acesso em: 30 jan. 2017.

MICROSOFT DEVELOPER NETWORK. Materialized View Pattern. Disponível em: <https://msdn.microsoft.com/pt-br/library/dn589782.aspx>. Acesso em: 20 fev. 2017.

MONGODB DOCUMENTATION. The MongoDB 3.4 Manual. Disponível em: <https://docs.mongodb.com/manual/>. Acesso em 28 fev. 2017.

SADALAGE, Pramod; FOWLER, Martin. **NoSQL Essencial:** Um Guia Conciso para o Mundo Emergente da Persistência Poliglota, São Paulo: Novatec Editora, 2013.

SATO, Priscila Mayumi. Banco de dados de Grafos. Disponível em: <https://pt.slideshare.net/MayogaX/banco-de-dados-de-grafos/>. Acesso em: 10 fev. 2017.

SATO, Priscila Mayumi. GraphDB Series: o que é um banco de dados de grafos. Disponível em: <https://imasters.com.br/banco-de-dados/graphdb-series-o-que-e-um-banco-de-dados-de-grafos/?trace=1519021197&source=single/>. Acesso em: 10 fev. 2017.

SILBERSCHATZ, Abrahan et. al. **Sistema de Banco de Dados**. 5.ed. Rio de Janeiro. Elsevier. 2006. p. 1-290.

SOUSA, Paulo. O teorema CAP. Disponível em: <https://unrealps.wordpress.com/2010/12/28/o-teorema-cap/>. Acesso em 27 fev. 2017.

STEPPAT, Nico. NoSQL – Do teorema CAP para P?(A/C): (C/L). Disponível em: <http://blog.caelum.com.br/nosql-do-teorema-cap-para-paccl/>. Acesso em 25 fev. 2017.

STRAPPAZZON, Nicola. El teorema CAP en base de datos. Disponível em: <https://www.swapbytes.com/teorema-cap-base-datos/>. Acesso em 30 mar. 2017.

VAISH, Gaurav. **Getting Started with NoSQL:** Your guide to the world and technology of NoSQL. Birmingham. Packt Publishing. 2013.

VIRTUABOX. Saiba mais sobre os clusters em armazenamento de arquivos. Disponível em: <https://virtuabox.wordpress.com/2009/12/09/saiba-mais-sobre-os-clusters-em-armazenamento-de-arquivos>. Acesso em: 28 jan. 2017.

WANSCHIK, Thomas. JOINs via denormalization for NoSQL coders, Part 2: Materialized views. Disponível em: <https://www.allbuttonspressed.com/blog/django/2010/09/JOINs-via-denormalization-for-NoSQL-coders-Part-2-Materialized-views>. Acesso em 22 fev. 2017.