UNIVERSIDADE FEDERAL FLUMINENSE

INSTITUTO DE COMPUTAÇÃO

ALBERTO MARTINEZ SCREMIN

BIANCA CARUSO DA PAIXÃO

ANALISE DE BANCO DE DADOS NÃO RELACIONAIS E COMPARAÇÃO COM BANCO DE DADOS RELACIONAIS

NITERÓI

2010

ALBERTO MARTINEZ SCREMIN

BIANCA CARUSO DA PAIXÃO

ANALISE DE BANCO DE DADOS NÃO RELACIONAIS E COMPARAÇÃO COM BANCO DE DADOS RELACIONAIS

Trabalho de Conclusão de Curso Apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal Fluminense para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: VANESSA BRAGANHOLO

NITERÓI

2010

ALBERTO MARTINEZ SCREMIN

BIANCA CARUSO DA PAIXÃO

ANALISE DE BANCO DE DADOS NÃO RELACIONAIS E COMPARAÇÃO COM BANCO DE DADOS RELACIONAIS

Trabalho de Conclusão de Curso Apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal Fluminense para obtenção do grau de Bacharel em Ciência da Computação.

Data da Aprovação: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

BANCA EXAMINADORA

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Prof. VANESSA BRAGANHOLO Orientador

UFF

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. <PROFESSOR 1>

UFF

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Prof. <PROFESSOR 2>

UFF

NITERÓI 2010

Dedicamos este trabalho aos nossospais que nos apoiaram durante todo o caminho traçado durante nossa vida acadêmica. Agradecemos também a professora Vanessa que nos orientou durante este trabalho.

AGRADECIMENTOS

Agradecemos, primeiramente, aos nossos pais que nos apoiaram durante todos os anos e em todas as fases. Devido a eles, nós tivemos condição de estudar e chegar até esta etapa de nossas vidas. Além disso, gostaríamos de agradecer aos nossos irmãos e familiares pelo carinho e apoio.

À nossa professora e orientadora, Vanessa Braganholo, pela paciência na orientação e incentivo que tornaram possível a conclusão desta monografia.

Aos funcionários da coordenação e da secretaria da UFF que nos acompanharam nesta jornada e estavam sempre dispostos a ajudar, em qualquer horário.

Aos nossos colegas de classe, que compartilharam diversos momentos, bons e ruins durante todos esses anos de atividade acadêmica.

RESUMO

Os SGBDs relacionais têm sido extremamente utilizados em praticamente todos os sistemas desenvolvidos nos últimos tempos. Estes oferecem muitas vantagens aos seus usuários, diversos mecanismos que permitem o controle da concorrência, segurança, integridade dos dados entre outros.

Entretanto, nos últimos anos, os bancos de dados não relacionais, também conhecidos como NoSQL ganharam um grande força no mercado devido ao crescimento do volume dos dados das organizações, além de outros fatores limitantes da estrutura pouco flexível dos modelos relacionais e principalmente pela questão da escalabilidade dos sistemas.

Tendo em vista este cenário, este documento propõe a análise de algumas características destes bancos relacionais e não relacionais e compará-los, tanto no aspecto de desempenho, quanto na facilidade desenvolvimento.

Palavras-chaves: Bancos de Dados, NoSQL, Modelo Relacional, Performance.

ABSTRACT

The relational database management systems have been greatly used in a lot of systems developed in recent times. They offer many advantages to its users, several mechanisms that allow control competition, security, data integrity among others.

However, in recent years, non-relational databases, also known as NoSQL won a major force in the market due to growth in the volume of data in organizations, and other factors such as the inflexibility of the structure of the relational models and especially the issue of scalability of the systems.

Having view this scenario, this paper proposes an examination of some characteristics of relational databases and the non-relational and comparethem both in terms of performance and ease development.

Keywords: Databases, NoSQL, Relational Databases, Performance

LISTA DE ILUSTRAÇÕES

[Figura 1: Exemplo de Modelo Relacional – Tabela Pessoa.15](#_Toc295923750)

[Figura 2: Comando SQL utilizado para criar tabelas no MySQL.17](#_Toc295923751)

[Figura 3: Comandos de inserção de dados nas tabelas MySQL.18](#_Toc295923752)

[Figura 4: Modelo de dados do Sedna21](#_Toc295923753)

[Figura 5: Criação de Indice no Sedna21](#_Toc295923754)

[Figura 6: Modelo de dados do Cassandra.23](#_Toc295923755)

[Figura 7: Criando uma keyspace.24](#_Toc295923756)

[Figura 8: Query utilizada para criar a estrutura da família de coluna Pessoa.24](#_Toc295923757)

[Figura 9: Comandos de inserção no Cassandra.24](#_Toc295923758)

[Figura 10: Comandos de consulta no Cassandra.24](#_Toc295923759)

[Figura 11: Comandos de remoção de uma linha e de uma família.24](#_Toc295923760)

[Figura 12: Esquema de um documento da coleção de pessoas.27](#_Toc295923761)

[Figura 13: Modelo de dados do Redis.30](#_Toc295923762)

LISTA DE ABREVIATURAS E SIGLAS

*ANSI*: *American National Standards Institute*

*BI: Business Intelligence*

*CLI: Command Line Interface*

*CSV*: *Comma-Separated Values*

*IP: Internet Protocol*

*ISO*:*International Standards Organization*

*NoSQL*: *Not Only SQL*

*OLTP: Online TransactionProcessing*

SGBD: Sistema de Gerenciamento de Banco de Dados

*SQL* :*Structured Query Language*

*SSL: SecureSockets Layer*

LISTA DE APÊNDICE

**Nenhuma entrada de índice remissivo foi encontrada.**

SUMÁRIO

[1 INTRODUÇÃO 14](#_Toc297044733)

[1.1 MOTIVAÇÃO 14](#_Toc297044734)

[1.2 OBJETIVO 14](#_Toc297044735)

[1.3 ORGANIZAÇÃO 14](#_Toc297044736)

[2 REVISÃO DA LITERATURA 15](#_Toc297044737)

[2.1 MYSQL 15](#_Toc297044738)

[2.2 SEDNA 20](#_Toc297044739)

[2.3 CASSANDRA 22](#_Toc297044740)

[2.4 MONGODB 25](#_Toc297044741)

[2.5 REDIS 28](#_Toc297044742)

[3 EXEMPLO DE APLICAÇÃO 33](#_Toc297044743)

[4 AVALIAÇÃO 33](#_Toc297044744)

[5 CONCLUSÃO 33](#_Toc297044745)

[6 REFERÊNCIAS BIBLIOGRÁFICAS 33](#_Toc297044746)

INTRODUÇÃO

* 1. MOTIVAÇÃO
  2. OBJETIVO
  3. ORGANIZAÇÃO

REVISÃO DA LITERATURA

* 1. MYSQL

Quando uma tecnologia é escolhida para ser utilizada em um sistema, existem alguns pontos que devem ser analisados como desempenho, condições e restrições da licença e preço. Sabendo disso, podemos analisar os recursos que o MySQL tem para oferecer (DUBOIS, 2000) :

* Velocidade
* Facilidade de administração e utilização
* Suporta SQL: é a linguagem mais utilizada para bancos de dados relacionais
* Capacidade: o MySQL é um servidor *multi-thread*, com isso vários clientes podem se conectar ao banco simultaneamente.
* Conectividade e Segurança.
* Portabilidade: o MySQL pode ser executado em diversas plataformas
* Disponibilidade e Custo: é um projeto *Open Source* disponível sobre diversos termos de licença, ou seja, pode ser executado sem nenhum custo ou através de acordos e compra de licença.
  + 1. Modelo de Dados

O MySQL é um banco de dados relacional (CODD, 1970), cujo modelo é formado por tabelas. Uma tabela é definida por um conjunto de atributos. Cada atributo pertence a um domínio (por exemplo, inteiro, data, etc). Desta forma, cada linha da tabela possui valores específicos para cada atributo da tabela. Uma linha da tabela também recebe o nome de tupla.

A Figura 1 mostra um exemplo da estrutura de uma tabela que representa uma entidade pessoa. Os atributos da tabela Pessoa são nome, CPF e data de nascimento.

|  |
| --- |
| tabela_pessoa |
| Figura 1: Exemplo de Modelo Relacional – Tabela Pessoa. |

As instâncias da tabela pessoa são representadas por suas tuplas. No exemplo da Figura 2 existem duas tuplas: <08539287409, João, 17-05-1967> e <05831765208, Maria, 21-10-1990>.



|  |
| --- |
|  |
| **Figura 2: Representação das tuplas no modelo relacional.** |

Os bancos de dados relacionais oferecem alguns recursos que facilitam a sua modelagem e o manuseio. São eles:

* Tipos de Dados: cada coluna pode ter um tipo de dado específico (texto, numérico, data, entre outros).
* Chaves CHECK: o SGBD pode restringir os valores que serão inseridos na coluna.
* Integridade Referencial: permite a criação de chaves referenciais, ou seja, não é possível criar um registro em uma coluna que referencie um dado inexistente na tabela pai.
* Junção: permite relacionar tabelas pelo valor de suas colunas.

Existe uma linguagem de consultas padrão para este tipo de Banco de Dados, que é a SQL.

Os bancos relacionais funcionam bem de acordo com as necessidades específicas do sistema, além de prover uma grande flexibilidade na criação de novas consultas, sem que estas estivessem planejadas no início da modelagem do projeto. Além disso, o banco relacional também funciona bem se a aplicação necessitar executar consultas complexas com muitas junções e necessitar da integridade referencial como parte das regras de negócio.

Entretanto, existem casos em que o banco de dados relacional não funciona com um bom desempenho. Supondo que uma aplicação precisa armazenar arquivos binários, ou os dados são estruturados como uma hierarquia ou um grafo. Neste caso é necessário utilizar uma topologia especial para realizar as principais funções da aplicação. Além disso, os bancos relacionais irão trazer mais complicações caso o sistema possua um modelo de dados que sofra alterações constantemente (STEPHENS, 2009).

* + 1. Linguagem de Consulta

O MySQL utiliza uma linguagem de consulta muito conhecida pela maioria dos administradores de banco de dados: o SQL (BEAULIEU, 2009). É uma linguagem bastante simples e proporciona uma grande facilidade de uso.

O SQL possui algumas subdivisões:

* DML – Linguagem de Manipulação dos dados – é utilizado para realizar inclusões, consultas, exclusões e alterações de dados presentes em registros. Os principais comandos são: *INSERT*, *UPDATE*e *DELETE.*
* DDL – Linguagem de Definição dos Dados – é utilizado para definir tabelas novas e elementos associados, ou para alterar uma tabela, adicionando uma nova coluna. Os principais comandos são: *CREATE, DROP, ALTER TABLE.*
* DCL – Linguagem de Controle de Dados – é utilizado para controlar os aspectos de autorização de dados e licenças de usuários visando a segurança da aplicação. Os principais comandos são: *GRANT, REVOKE.*
* DTL – Linguagem de Transação de Dados – é usado para marcar o começo de uma transação de banco de dados que pode ser completada ou não, por exemplo: *START TRANSACTION.* Além disso, existe também o comando *COMMIT* que realiza todas as mudanças dos dados permanentemente, e o *ROLLBACK,* que faz com que as mudanças nos dados existentes desde o último *COMMIT* ou *ROLLBACK* sejam descartadas.
* DQL – Linguagem de Consulta de Dados – é utilizada para realizar uma consulta na base de dados de forma a encontrar um resultado específico. O único comando desta subdivisão é o *SELECT*.

Embora o SQL tenha sido padronizado pela ANSI e ISO (ANSI/ISO, 1986) na década de 80, a linguagem possui muitas variações e extensões produzidas pelos diferentes fabricantes de sistemas gerenciadores de bases de dados. Sendo assim, estas subdivisões e os comandos podem sofrer alterações dependendo do SGBD.

Na Figura 3 e na Figura 4 são apresentados respectivamente alguns comandos utilizados para criar a estrutura e os dados da tabela Pessoa (Figura 1).

|  |
| --- |
| CREATE TABLE `pessoa` (  `cpf` bigint(20) unsigned NOT NULL,  `nome` varchar(100) default NULL,  `data\_nascimento` date default NULL,  PRIMARY KEY(`cpf`)  ) ; |

Figura 3: Comando SQL utilizado para criar tabelas no MySQL.

|  |
| --- |
| INSERT INTO pessoa (cpf, nome, data\_nascimento) VALUES (08539287409, 'Joao', '1967-05-17');  INSERT INTO pessoa (cpf, nome, data\_nascimento) VALUES (05831765208, 'Maria', '1990-10-21');  COMMIT; |

Figura 4: Comandos de inserção de dados nas tabelas MySQL.

* + 1. Índices

Os índices são os principais meios de acelerar o acesso ao conteúdo das tabelas, especialmente se a consulta envolver múltiplas junções entre as tabelas. Sem o índice, o MySQL é obrigado a ler a tabela por completo, desde a primeira linha até a última linha da tabela. Quanto maior for esta tabela, maior será o custo da consulta. Se a tabela possui índice sobre a coluna em questão, ou seja, nas colunas que são referenciadas na cláusula *WHERE*, o MySQL consegue facilmente determinar a posição que deve procurar no arquivo de dados, sem ter que acessar todos os dados. Entretanto, se a consulta requerer como resposta a maioria das linhas da tabela, é mais rápido ler o arquivo de dados sequencialmente, minimizando acessos ao disco.

A maioria dos índices do MySQL são armazenados em Árvores-B, como por exemplo os índices *Primary Key, Unique Key, Index e Fulltext*. Entretanto, alguns tipos específicos de tabelas também suportam *Hash Indexes*.

O MySQL é bem flexível na construção dos índices. É possível indexar uma única coluna, ou uma combinação de colunas. Além disso, é possível ter mais de um índice por tabela.

O número máximo de índices por tabela e o tamanho máximo de um índice varia de acordo com a forma de armazenamento que foi adotado na criação da tabela. Todas elas suportam no mínimo 16 índices por tabela. Se a coluna for do tipo string, é possível indexar somente os *n* primeiros caracteres da coluna, tornando o índice menor e mais rápido. No MySQL, existe uma forma de armazenamento denominada MyISAM. Nesta é possível criar *fulltext* índices que são utilizados para consultas no texto inteiro.

Em alguns casos o MySQL não utiliza o índice, mesmo se possuir algum disponível. Um caso em que isso ocorre é quando o otimizador estima que a utilização do índice irá requerer muitos acessos ao disco. Isso significaria em um grande percentual de linhas da tabela sendo lido. Neste caso, ler a tabela por completo é mais rápido. Entretanto, se a mesma consulta utilizar o LIMIT, comando que limita o tamanho do resultado em um subconjunto de linhas, o MySQL utiliza o índice.

Um exemplo de criação de índice no MySQL pode ser visto a seguir. Neste comando é criado um índice chamado id\_nome sobre o campo nome da tabela pessoa.

|  |
| --- |
| **CREATE INDEX id\_nome ON pessoa(nome);** |

O MySQL possui um otimizador que provê informações do plano de execução das consultas. Além disso, descreve como realizar a configuração dos *caches* das chaves para armazenar informações dos índices em memória.

Quando uma consulta não executa tão rápido quanto se gostaria, é possível utilizar a cláusula *EXPLAIN* para pedir ao servidor MySQL informações sobre como o otimizador de consulta processa a consulta. Esta informação pode ser útil por diversos motivos:

* *EXPLAIN* pode informar pontos da consulta que precisam de índices;
* Se a tabela já possui um índice, o *EXPLAIN* mostra se o otimizador está utilizando o mesmo;
* Se o índice existe mas não está sendo utilizado, o analista pode reescrever a consulta de diversas maneiras diferentes. O *EXPLAIN* pode dizer qual das consultas é a melhor para ajudar o servidor a utilizar os índices disponíveis.

Para utilizar este mecanismo de otimização, basta escrever a consulta normalmente e colocar a cláusula *EXPLAIN* no início da consulta. Em seguida, é mostrado um exemplo de duas consultas feitas na tabela Pessoa, descrita na Figura 1. Assumindo que foram inseridos 670 registros na base de dados, e que e existe apenas um João, cujo CPF é 08539287409, as duas consultas retornam o mesmo resultado. Porém, elas não são igualmente eficientes.

|  |
| --- |
| **mysql> explain select \* from pessoa where nome = 'Joao'; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **id: 1**  **select\_type: SIMPLE**  **table: pessoa**  **type: ALL**  **possible\_keys: NULL**  **key: NULL**  **key\_len: NULL**  **ref: NULL**  **rows: 506**  **Extra: Using where**  **1 row in set (0.00 sec)**  **mysql> explain select \* from pessoa where cpf = 08539287409;**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **id: 1**  **select\_type: SIMPLE**  **table: pessoa**  **type: const**  **possible\_keys: PRIMARY**  **key: PRIMARY**  **key\_len: 8**  **ref: const**  **rows: 1**  **Extra:**  **1 row in set (0.00 sec)** |

O comando *explain* retorna diversas informações. Na primeira consulta, a palavra *NULL* nas colunas *possible\_keys* e *key* mostram que nenhum índice foi considerado disponível e também nenhum foi utilizado para realizar a consulta. Já na segunda consulta, o índice da chave primária da tabela é proposto como um possível índice a ser utilizada, e é exatamente o índice que o otimizador irá escolher para trazer os resultados.

A coluna *rows* indica os efeitos desta diferença, ou seja, o valor indica a quantidade de linhas que o MySQL estima que serão analisadas enquanto processa a consulta. A primeira consulta retorna o valor 506, que é grande parte da tabela Pessoa, ou seja, o MySQL irá processar quase todas as linhas da tabela antes de encontrar o registro de João, mostrando que a consulta é extremamente ineficiente. Já na segunda consulta somente uma linha é processada já que o MySQL utiliza o índice da chave primária, conseguindo ir direto para a única linha realmente relevante da consulta (DUBOIS, 2008).

* 1. SEDNA

Sedna é um banco de dados nativo XML (BRAY et al., 2008), open source, que implementa alguns serviços comuns aos bancos de dados relacionais, como: armazenamento persistente, transações ACID, segurança, índices. Suas consultas são realizadas em XQuery (BOAG et al., 2010), ou através do XPath (CLARK; DEROSE, 2003).

Sedna possui como objetivo prover um ambiente em tempo de execução para aplicações que processam dados XML. Isso implica uma estreita integração das funcionalidades da gestão de banco de dados com a linguagem de programação (CUONG, 2006).

* + 1. Modelo de Dados

No Sedna, o modelo de dados utilizado é o XML. XML é uma linguagem poderosa para a representação de dados semi-estruturados. Dados semi-estruturados são aqueles que contêm seus esquemas embutidos dentro dos dados. Dessa forma, se auto-descrevem (BUNEMAN, 1997).

XML é simples, flexível e derivado do SGML. Foi projetado para superar o desafio da publicação de arquivos em larga escala, mas atualmente tem um papel grande na troca de informações na Internet (BRAY et al., 2008). Um documento XML tem que seguir algumas especificações para que seja bem formado. São elas:

* Devem possuir uma raiz única;
* Não possuem marcas abertas;
* Elementos têm que ser bem aninhados;
* Atributos são únicos por elementos;
* Elementos são sensíveis a maiúsculas e minúsculas (BRAY et al., 2008).

Na Figura 5 temos o exemplo de um arquivo XML bem formado.

|  |
| --- |
| <pessoas>  <pessoa>  <cpf>08539287409</cpf>  <nome>Joao</nome>  <data\_nascimento>1967-05-17</data\_nascimento>  </pessoa>  <pessoa>  <cpf>05831765208</cpf>  <nome>Maria</nome>  <data\_nascimento>1990-10-21</data\_nascimento>  </pessoa>  </pessoas> |

Figura 5: Exemplo de Documento XML

* + 1. Linguagem de Consulta

XQuery é uma linguagem de consultas padrão da W3C (BOAG et al., 2010) utilizada pelo Sedna para a manutenção de seus dados. Utilizando-se da mesma estrutura de um XML, a XQuery permite consultas a todos elementos de documentos baseados em XML.

Um exemplo de consulta para que sejam retornados todos os nomes das pessoas do documento da Figura 5 seria expresso da seguinte forma:

**doc("pessoas.xml")/pessoas/nome**

* + 1. Índices

Sedna utiliza de árvores B para armazenar seus índices. Os índices permitem que um elemento seja indexado pelo valor de um de seus subelementos. Como um exemplo, na Figura 6 o CPF está sendo usando para indexar os elementos pessoa do documento XML da Figura 5.

O primeiro caminho descrito na Figura 6 (doc(“pessoas”)/pessoas/pessoa), expresso por uma expressão XPath, indica qual elemento vai ser retornado por esse índice. O segundo caminho (cpf), também expresso por uma expressão XPath, indica o campo que vai ser indexado. Nota-se que o tipo desse campo deve ser escrito explicitamente.

|  |
| --- |
| CREATE INDEX “cpf” ON doc(“pessoas”)/pessoas/pessoa BY cpf AS xs:string |

Figura 6: Criação de índice no Sedna

Entretanto, o Sedna não utiliza índices por padrão em suas consultas XQuery. É necessário a utilização de uma função, conhecida como *index-scan*,para que o Sedna utilize o índice criado anteriormente (Institute for System Programming RAS, 2003).

O Sedna apresenta algumas camadas para otimização de consultas baseadas em regras. O Sedna utiliza-se de seis regras para otimização de consultas (GRINEV et al., 2004):

* Técnica de aninhamento de funções: Troca as chamadas das funções pelas funções em si, tornando as outras técnicas de otimização mais fáceis de serem implementadas;
* Adiamento da construção de elementos XML em predicados: através desta técnica, a tarefa de construção de elementos XML, que tem alto custo, é feita apenas após o resultado já ter sido filtrado. Utiliza-se da projeção da transformação, com expressões XPath para construção dos elementos, reduzindo assim o custo de operações intermediárias, com a retirada de operações redundantes;
* Simplificação de consultas através de esquema associado: permite que o usuário que não conheça o esquema associado ao documento que está sendo consultado consiga um melhor desempenho em suas consultas. Essa técnica é baseada na inferência estática da XQuery, criando consultas mais precisas;
* Fazer com que consultas sejam o mais declarativas possíveis: permite que o otimizador possa procurar por um conjunto maior de planos execução de consultas;
* Normalização de junções: consiste em reescrever as consultas de forma que possa ser aplicada não apenas iterações aninhadas, mas também outros algoritmos de junções. O Sedna consegue isso através da extração de expressões XPath de dentro das junções;
* Identificação de operações livres de iteração dentro do corpo de iterações: o que reduz o custo das consultas retirando estes tipos de ocorrência.
  1. CASSANDRA

O Cassandra é um SGBD *open source* não relacional. Apesar de muitos o denominarem como um banco de dados orientado a colunas, ele pode ser encarado como um índice. Isso porque o Cassandra utiliza o armazenamento dos dados baseado em linhas, onde cada linha possui uma chave única que torna o dado acessível . Sabendo disso, podemos classificar o Cassandra como um Banco de Dados orientado a Chave-Valor (HEWITT, 2011).

* + 1. Modelo de Dados

O modelo de dados do Cassandra funciona ligeiramente diferente dos modelos relacionais. Foram criados alguns conceitos completamente novos, como o *keyspace*. Entretanto, também existem conceitos que pertencem ao modelo relacional e ao modelo chave-valor, porém com significados diferentes, como é o caso das colunas.

O conceito do *keyspace* é semelhante ao do banco de dados no modelo relacional. No modelo relacional, o banco é o repositório das tabelas e assim como ele, o *keyspace* é considerado o recipiente de pelo menos uma família de colunas. Cada *keyspace* possui um nome e uma lista de atributos que definem o seu comportamento.

Uma família de coluna (*column family*) é análoga às tabelas no modelo relacional. Ela representa a estrutura dos dados e é considerada o repositório para as coleções de linhas. Estas linhas são formadas por um conjunto de colunas, mas este conjunto não é necessariamente igual para todas as linhas (HEWITT, 2011).

O modelo da Figura 7 mostra um exemplo da família de coluna Pessoa, composta por duas linhas, onde a chave da linha é o CPF. Cada linha possui duas colunas, que são os atributos nome e data de nascimento.

|  |
| --- |
| modelo_cassandra |
| Figura 7: Modelo de dados do Cassandra. |

Podemos representar os dados deste modelo (Família Pessoa) como é mostrado abaixo:

|  |
| --- |
| **Pessoa:**  **08539287409:**  **Nome: Joao**  **DataNascimento: 1967-05-17**  **05831765208:**  **Nome: Maria**  **DataNascimento: 1990-10-21** |
|  |

* + 1. Linguagem de Consulta

Existem algumas diferenças entre as consultas realizadas no Cassandra e nos bancos relacionais, que são explicadas a seguir (HEWITT, 2011).

* O Cassandra não realiza atualizações de registros, ou seja, não existe um comando de *update*. Para atualizar um valor, basta inseri-lo em uma chave já existente. Sempre que isso ocorre, o Cassandra substituiu o dado existente pelo dado que foi inserido.
* O Cassandra não possui suporte a transações
* Não permite chaves duplicadas, ou seja, se uma chave que já existe no banco tentar ser inserida novamente, os dados cadastrados serão sobrescritos.
* As escritas no Cassandra são muito rápidas, pois foi implementado para não realizar idas ao disco durante este processo.

Nos exemplos das Figura 8 e Figura 9 é possível verificar exemplos de criação da keyspace e da família de coluna, respectivamente. Na Figura 10, Figura 11 e Figura 12, são representados os comandos de inserção de dados, consultas e remoção de dados, respectivamente. Todos os comandos foram executados no *CLI*, um *client* do Cassandra.

|  |
| --- |
| Create create keyspace usuarios; |

Figura 8: Criando uma keyspace.

|  |
| --- |
| **create column family pessoa with comparator=UTF8Type**  **and column\_metadata=[{column\_name: nome, validation\_class: UTF8Type},**  **{column\_name: data\_nascimento, validation\_class: LongType}];** |

Figura 9: Query utilizada para criar a estrutura da família de coluna Pessoa.

|  |
| --- |
| **set pessoa[**08539287409**][nome] = 'Joao';**  **set pessoa[**08539287409**][ data\_nascimento] =** 1967-05-17**;**  **set pessoa[**05831765208**][nome] = 'Maria';**  **set pessoa[**05831765208**][ data\_nascimento] =** 1990-10-21**;** |

Figura 10: Comandos de inserção no Cassandra.

|  |
| --- |
| **get pessoa where nome = 'Joao';**  **get pessoa wheredata\_nascimento = 1967-05-17;** |

Figura 11: Comandos de consulta no Cassandra.

|  |
| --- |
| del pessoa[08539287409]['nome];  delusers[08539287409]; |

Figura 12: Comandos de remoção de uma linha e de uma família.

No Cassandra, podemos criar uma coluna com um tempo de expiração, ou seja, dado este tempo, a coluna é apagada. Para isto, basta colocar no comando de *set* a seguinte cláusula: *withttl*=<segundos>;

* + 1. Índices

O Cassandra possui um índice que é criado automaticamente, o índice da chave primária. Este é criado na chave de linha de todas as famílias de coluna. A última versão lançada do Cassandra – versão 0.7 – passou a suportar índices secundários (índices nas colunas) que são índices *hash* e permitem consultas por qualquer valor. Além disso, podem ser construídos em segundo plano automaticamente, sem bloquear leituras e escritas. Entretanto, esse tipo de índice possui uma série de limitações, são elas (ELLIS, 2010):

* Por ser um índice *Hash*, não permite a consulta por um subconjunto de valores, o índice só é utilizado em consultas de igualdade.
* É recomendado que se utilizem índices em atributos que possuem baixa cardinalidade.

Para adicionar um índice na coluna nome, basta utilizar o comando a seguir.

|  |
| --- |
| **update column family pessoa with comparator=UTF8Type**  **and column\_metadata=[ {column\_name: nome, validation\_class: UTF8Type, index\_type: KEYS },{column\_name: data\_nascimento, validation\_class: LongType }];** |

O Cassandra utiliza as estatísticas para a escolha dos índices que serão usados na consulta, de forma que irá ler a menor quantidade de linhas possíveis.

* 1. MONGODB

MongoDB é um banco de dados que faz a utilização de coleções ao invés de tabelas como nos bancos relacionais e da utilização de documentos que podem ser vistas como as linhas dos bancos relacionais. Dessa forma ele é classificado como um Banco Orientado a Documentos.

* + 1. Modelo de Dados

No MongoDB não existem junções entre as coleções, que seriam as tabelas de um banco de dados relacional. Naturalmente, não se deve desnormalizar a maior parte do banco, pois não se pode contar com as junções. No lugar, o que deve ser utilizado, são os documentos que contém documentos, levando assim a não termos uma coleção para cada objeto da aplicação.

Diz-se que a questão principal do MongoDB consiste em escolher os objetos que devem virar coleções. Deve-se levar em conta algumas das considerações gerais como:

* Objetos de primeira classe devem tornar-se coleções;
* Relacionamentos muitos para muitos geralmente se relacionarão por referência;
* Coleções com poucos objetos podem existir como coleções separadas;
* Documentos têm limite de 8MB, o que pode ser um limite para documentos conterem outros documentos.

Os documentos também podem ser referenciados, para normalização e contar com alguma integridade. Essa referência pode ser feita de duas formas: manualmente ou usando *dbref*. Sendo que manualmente, o desenvolvedor teria que tratar essa referência com uma consulta a mais no banco. Além da forma manual, o MongoDB permite uma forma de referenciar outros documentos de uma maneira mais formal, o *DBRef*. *DBRefs* nada mais são que documentos contidos no próprio documento e que geralmente incluem o nome da coleção com um *ObjectID*. Desenvolvedores preferem utilizar o *DBRef* quando as coleções referenciadas diferem de um documento para outro, caso contrário a referência manual é mais eficiente (MERRIMAN; CHODROW, 2011). Cabe lembrar que para cada referência é necessária uma consulta na coleção referenciada para o retorno do objeto. Isso pode ter um alto custo para o desempenho caso a coleção referenciada seja muito grande. Para a criação desse tipo de referência é necessário a passagem de dois argumentos, da seguinte forma:

**{ $ref : <nome da coleção>, $id : <valor do \_id>[, $db : <nome do database>] }**

Outra característica do MongoDB é que ele pode conter qualquer tipo de documento dentro de suas coleções, ou seja, suas coleções não têm um esquema definido, os objetos podem ser completamente diferentes uns dos outros dentro da mesma coleção. Apesar de possível e de ser utilizada por alguns bancos de dados, essa estratégia não é recomendada por razões de processamento.

Como exemplo de esquema de um banco é o caso de uma aplicação de um blog. Onde os autores e artigos seriam coleções e os comentários referentes aos artigos tornar-se-iam documentos inseridos nos documentos pertencentes à coleção de artigos(MURPHY; VOYER-PERRAULT, 2011).

Considerando o exemplo da coleção pessoa, o MongoDB não precisa de nenhum comando para criá-la, por ser um banco livre de esquema. Na Figura 13 é mostrado como ficaria nossa coleção contendo os documentos com informações das pessoas.

|  |
| --- |
|  |
| Figura 13: Esquema de um documento da coleção de pessoas. |

* + 1. Linguagem de Consulta

A linguagem de consulta utilizada pelo MongoDB é o BSON. BSON são documentos no formato binário do JSON. Esse documento BSON é gerado pelo MongoDB através dos documentos JSON enviados pelo cliente. Frequentemente as consultas que poderiam ser feitas com o SQL, podem também ser expressas com o JSON (CHODROW; GILL, 2010).

JSON é um tipo de documento derivado do JavaScript para representação de estrutura de dados e objetos. Sendo assim é uma linguagem de marcação, como o XML. Apesar de não ser extensível como o XML, o JSON é considerado uma linguagem mais legível para humanos e mais eficiente para máquinas (ECMA INTERNATIONAL, 2006).

O documento JSON é construído sobre as seguintes estruturas:

* Coleção de pares chave/valor;
* Uma lista de valores ordenada.

Como exemplo de um documento JSON, temos:

**{“cpf” : “129304229”, “nome”: “João”, “data\_nascimento” : Date(‘1967,05,17’)}**

São necessários modificadores de consultas para fazermos consultas por critérios. Esses modificadores podem ser operadores condicionais, ou mesmo operações como *in*, *exists*, dentre outras. São operadores que são facilmente resolvidos pelo banco, pois são tratados como parte do documento BSON.

Mas, se é necessária uma consulta que não seria possível utilizando os modificadores, também é possível a utilização de consultas com *map/reduce* (DEAN; SANJAY, 2008). *Map/reduce*, é uma ferramenta poderosa de processamento de dados e para agregações. Geralmente complicadas de se escrever, são utilizadas quando seria utilizada uma função de *group by* em SQL. No MongoDB, funciona da seguinte forma: é selecionada uma coleção, processada segundo a função de *map/reduce* e então retorna uma nova coleção com o resultado.

Apesar de o map/reduce ser uma ferramenta poderosa, ela apresenta um desempenho ruim para resultados em tempo real. Devendo-se ao fato da limitação do motor do JavaScript, o *map/reduce* pode ser apenas executado em uma *thread*, logo não aproveita os vários núcleos que os processadores atuais possuem. Além de não permitir que outros processos sejam executados em paralelo (HOROWITZ; STEARN, 2011).

* + 1. Índices

Os índices no MongoDB funcionam de maneira muito semelhante aos índices nos banco relacionais. Logo a maioria dos conceitos e materiais sobre indexação para bancos relacionais também podem ser utilizados com o MongoDB (CHODROW; DIROLF, 2010).

O índice é uma estrutura de dados que coleta informações sobre o valor de um determinado campo do documento, implementada em Árvore-B e utilizada pelo otimizador de consultas do MongoDB para rapidamente ordenar os documentos da coleção (MURPHY; MERRIMAN, 2011).

Quando uma única chave é utilizada na consulta, esta chave pode ser indexada para melhorar o desempenho da consulta. Entretanto, se o campo não está sendo chamado no método *find*, o índice não irá melhorar a desempenho da consulta. Para criar um índice no MongoDB, basta utilizar o método *ensureIndex*, indicar quais campos serão indexados e indicar se o índice é ascendente (com o parâmetro 1) ou descendente (com o parâmetro -1). Para adicionar um índice no campo cpf do documento da Figura 13, devemos utilizar o seguinte comando:

**db.pessoas.ensureIndex({‘cpf’ : 1});**

O MongoDB suporta chaves únicas, índices compostos, índices em qualquer campo do documento, inclusive nos documentos inseridos e índices geoespaciais. O MongoDB, também permite índices esparsos, permitindo assim consultas utilizando o índice em coleções onde nem todos documentos possuem o campo indexado, tendo apenas como limitação não possuir mais de um campo indexado

Índices também podem ser utilizados limitar o resultado de uma consulta ao restringir os campos da consulta para apenas os campos que possuem determinado índice. Outra característica do MongoDB é que um índice único, por padrão, é criado no campo ‘\_id’. Esse índice apenas não será criado se a coleção for do tipo *capped,* que é um tipo especial de coleção.

A desvantagem de criar índices é que este causa uma sobrecarga nas inserções, modificações e exclusões. Isto se deve ao banco além da operação, precisar também alterar os índices, por isso não é recomendada a indexação de todos os campos da coleção.

O otimizador de consultas do MongoDB gera planos para todas as consultas que forem submetidas por um cliente, funcionando de uma forma diferente dos otimizadores tradicionais, que se baseiam no custo das consultas. Ao contrário destes modelos, o otimizador simplesmente executa diferentes planos em paralelo e aprende qual deles trabalha melhor, que é aquele que terminar primeiro. Este método funciona bem para bancos não relacionais, pois não são realizadas junções, logo o espaço utilizado para os possíveis planos da consulta é muito menor.

Pode acontecer de um plano que funcionava bem, com o tempo apresentar um desempenho pior, com a consulta demorando mais a responder. O motivo dessa queda de desempenho ocorre devido a mudanças nos dados do banco ou mudanças nos valores dos parâmetros das consultas. Neste caso, o SGBD executará novamente diferentes planos em paralelo para verificar qual o mais adequado. (MERRIMAN, 2010)

Outra característica do MongoDB é que caso seja necessário, o otimizador de consultas irá reordenar os termos da consulta para obter as vantagens do índice. Por exemplo, caso a consulta seja { “x”:“foo”, “y”:“bar”} e existe o seguinte índice {“y”:1, “x”:1}, o MongoDB irá resolver utilizar o índice (CHODROW; DIROLF, 2010).

* 1. REDIS

Redis é um banco de dados de execução de um dicionário, onde cada chave é associada a um valor. O que faz do Redis um banco diferente de muitos outros também orientados a chave-valor, é que todo valor singular tem um tipo. Além disso, o Redis é um banco muito rápido pois ele carrega e armazena todos os dados em memória e em segundo plano as alterações são escritas em disco.

* + 1. Modelo de Dados

O Redis é um modelo orientado a chave-valor. A essência deste modelo é a habilidade de armazenar dados, chamados de valor dentro de uma chave, como mostra a Figura 14. Este dado pode ser resgatado somente pela chave exata onde foi armazenado, ou seja, não existe uma busca por valor. O Redis funciona como um dicionário persistente. Todo valor armazenado possui um tipo (FINDLEY, 2010). Os tipos suportados são:

* Texto
* Lista
* Conjunto (*Set*)
* ConjuntoOrdenado (*Sorted Set*)

O tipo de um valor determina quais operações (chamados de comandos) estão disponíveis para o valor em si (SANFILIPPO; NOORDHUIS, 2010). Por exemplo, para acrescentar elementos para uma lista armazenada na chave *x* é utilizado o comando LPUSH ou RPUSH. Na seção seguinte serão mostrados os comandos da linguagem. Cada comando é realizado do lado do servidor através de operações atômicas.

|  |
| --- |
| modelo_redis |
| Figura 14: Modelo de dados do Redis. |

* + 1. Linguagem de Consulta

O Redis possui um *client* de consulta nativo, chamado redis-cli. Este é instalado junto com o banco de dados. Como foi dito anteriormente para cada tipo de valor existe uma operação específica que deve ser executada no banco. Abaixo serão mostrados alguns exemplos mais utilizados (SANFILIPPO; NOORDHUIS, 2010).

* Comandos gerais:
  + *Exists* <chave> – Testa se uma chave existe.
  + *Del* <chave> – Apaga uma chave.
  + *Type* <chave> – Retorna o tipo do valor armazenado na chave.
  + *Key* <padrão> – Retorna todas as chaves que possuem o padrão passado.
  + *Dbsize* – Retorna a quantidade de chaves presentes no banco.
  + *Flushdb* – Remove todas as chaves do banco de dados corrente.
* Tipo Texto:
  + *Set* <chave> <valor> - Atribui um valor texto a uma chave.
  + *Get* <chave> - Retorna o campo texto da chave pesquisada.
  + *Getset* <chave> <valor> - Atribui um valor a uma chave e retorna o valor antigo da chave.
  + *Setnx* <chave> <valor> - Atribui um valor a uma chave se a chave não existir no banco.
* Tipo Lista:
  + *Rpush* <chave> <valor> - Adiciona um valor ao final da lista de uma chave.
  + *Lpush* <chave> <valor> - Adiciona um valor ao início da lista de uma chave.
  + *Llen* <chave> - Retorna o tamanho da lista de determinada chave.
  + *Lindex* <chave> <índice> - Retorna o elemento da lista na posição do índice citado.
  + *Lrange <chave> <inicio> <fim> -* Retorna um subconjunto de elementos de uma lista.
  + *Lpop* <chave> - Retorna e remove o primeiro elemento da lista.
  + *Rpop* <chave> - Retorna e remove o ultimo elemento da lista.
* Tipo Conjunto:
  + *Sadd* <chave> <membro> - Adiciona o membro ao valor do conjunto.
  + *Spop* <chave> <membro> - Remove e retorna o membro específico do valor do conjunto.
  + *Sunion* <chave1> <chave2>...<chave *n*> - Retorna a união entre os conjuntos armazenados nas chaves 1, chave2,..., chave *n*.
  + *Smembers* <chave> - Retorna todos os membros do conjunto de uma determinada chave.
* Tipo Conjunto Ordenado:
  + *Zrem* <chave> <membro> - Remove o membro do conjunto
  + *Zadd* <chave> <posição> <membro> - Adiciona um membro ao conjunto. Caso já exista, somente a posição é alterada
  + *Zrange* <chave> <inicio> <fim> - Retorna um subconjunto dos elementos do conjunto ordenado.
* Ordenação:
  + *Sort* <chave> [*by* <padrão>][*limit* <quantidade>][*asc*|*desc*][*alpha*] – Ordena os elementos da lista, do conjunto ou do conjunto ordenado de determinada chave. Todas as cláusulas entre colchetes são opcionais.
* Comandos de persistência:
  + *Save* - Salva o banco em disco, de forma síncrona.

A seguir, estão representados alguns destes comandos, baseados no esquema montado na Figura 14.

|  |
| --- |
| **>rpush cpf:08539287409 nome:Joao**  **>rpush cpf:08539287409 data\_nascimento:1967-05-17**  **>lrange cpf:08539287409 0 2**  **1) "nome:Joao"**  **2) "data\_nascimento:1967-05-17"**  **>rpush cpf:05831765208 nome:Maria**  **>rpush cpf:05831765208 data\_nascimento:1990-10-21**  **>lrange cpf:05831765208 0 2**  **1) "nome:Maria"**  **2) "data\_nascimento: 1990-10-21"** |

* + 1. Índices

O Redis não suporta a criação de índices. O usuário é o responsável por criar os índices manualmente (RUSSO, 2010).

Ao utilizar o Redis, deve-se lembrar que a organização dos dados deve ser feita de acordo com as consultas que serão realizadas. O Redis não implementa um otimizador de consultas. Ele provê primitivas muito rápidas. Entretanto, o desempenho da consulta é altamente dependente de como o usuário opta por organizar os dados (RUSSO, 2010).

* 1. Considerações Finais

No capítulo 2 foram vistas as características de modelo dos dados, como são expressas as consultas, criação de índices e como as consultas são otimizadas em cada um dos bancos que serão utilizados no próximo capítulo.

No capítulo 3 será abordado um esquema comum, retirado de uma ferramenta de *benchmark* e algumas consultas para mostrar o poder de expressão de cada um dos bancos selecionados e também a dificuldade para conseguir fazê-las.

EXEMPLO DE APLICAÇÃO

* 1. TPC-E

AVALIAÇÃO

* 1. DESEMPENHO
  2. MODELAGEM

CONCLUSÃO

REFERÊNCIAS BIBLIOGRÁFICAS

ANSI/ISO. **SQL-86**, 1986.

BEAULIEU, A. **Learning SQL**. 2nd ed. Beijing ;;Sebastopol: O’Reilly, 2009.

BOAG, S.; CHAMBERLIN, D.; FERNANDEZ, M. F.; et al. XQuery 1.0: An XML Query Language. . Retrieved May 17, 2011, from http://www.w3.org/TR/xquery/, 2010.

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M.; MALER, E.; YERGEAU, F. Extensible Markup Language (XML) 1.0 (Fifth Edition). . Retrieved May 24, 2011, from http://www.w3.org/TR/REC-xml/, 2008.

BUNEMAN, P. Semistructured data. **ACM SIGACT-SIGMOD-SIGART**, PODS ’97., p. 117–121, 1997.

CHODROW, K.; DIROLF, M. **MongoDB: The Definitive Guide**. 1st ed. O’Reilly Media, 2010.

CHODROW, K.; GILL, M. Querying - MongoDB. . Retrieved May 12, 2011, from http://www.mongodb.org/display/DOCS/Querying, 2010.

CLARK, J.; DEROSE, S. XML Path Language (XPath). . Retrieved June 9, 2011, from http://www.w3.org/TR/xpath/, 2003.

CODD, E. F. A Relational Model of Data for Large Shared Data Banks. ,1970.

CUONG, N. XML Native Database Systems Review of Sedna, Ozone, NeoCoreXMS. ,2006.

DEAN, J.; SANJAY, G. MapReduce: Simplified Data Processing on Large Clusters. , v. Communications of the ACM 51(1): 107-113, 2008., 2008.

DUBOIS, P. **MySQL**. Indianapolis IN: New Riders Pub., 2000.

DUBOIS, P. **MySQL 5.0 Certification Study Guide**. 1st ed. Indianapolis Ind.: MySQL Press, 2008.

Ecma International. JSON: The Fat-Free Alternative to XML. . Retrieved May 12, 2011, from http://www.json.org/xml.html, 2006.

ELLIS, J. Cassandra Developer Center - What’s new in Cassandra 0.7: Secondary indexes. . Retrieved May 17, 2011, from http://www.datastax.com/dev/blog/whats-new-cassandra-07-secondary-indexes, 2010, March 12.

FINDLEY, R. Redis Overview - SlideShare. . Retrieved May 19, 2011, from http://www.slideshare.net/neomindryan/redis-overview-presentation, 2010.

GRINEV, M.; FORMICHEV, A.; KUZNETSOV, S. Sedna: A Native XML DBMS. ,2004.

HEWITT, E. **Cassandra : the definitive guide**. Beijing: O’Reilly, 2011.

HOROWITZ, E.; STEARN, M. MapReduce - MongoDB. . Retrieved May 17, 2011, from http://www.mongodb.org/display/DOCS/MapReduce, 2011.

Institute for System Programming RAS. **Sedna Programmer’s Guide**, 2003.

MERRIMAN, D. Query Optimizer - MongoDB. . Retrieved May 13, 2011, from http://www.mongodb.org/display/DOCS/Query+Optimizer, 2010.

MERRIMAN, D.; CHODROW, K. Database References - MongoDB. . Retrieved May 16, 2011, from http://www.mongodb.org/display/DOCS/Database+References, 2011.

MURPHY, R.; MERRIMAN, D. Indexes - MongoDB. . Retrieved May 12, 2011, from http://www.mongodb.org/display/DOCS/Indexes, 2011.

MURPHY, R.; VOYER-PERRAULT, G. Schema Design - MongoDB. . Retrieved May 9, 2011, from http://www.mongodb.org/display/DOCS/Schema+Design, 2011.

RUSSO, M. J. Redis, from the Ground Up. . Retrieved May 23, 2011, from http://blog.mjrusso.com/2010/10/17/redis-from-the-ground-up.html, 2010.

SANFILIPPO, S.; NOORDHUIS, P. Command reference – Redis. . Retrieved May 19, 2011, from http://redis.io/commands, 2010.

STEPHENS, R. **Beginning Database Design Solutions**. Indianapolis IN: Wiley Pub., 2009.