UNIVERSIDADE PAULISTA – UNIP

Guilherme de Pontes

ANÁLISE E COMPARAÇÃO DE DESEMPENHO ENTRE SGBD RELACIONAL E NÃO-RELACIONAL

Limeira

2017

UNIVERSIDADE PAULISTA – UNIP

Guilherme de Pontes

ANÁLISE E COMPARAÇÃO DE DESEMPENHO ENTRE SGBD RELACIONAL E NÃO-RELACIONAL

Trabalho de conclusão de curso apresentado à banca examinadora da Faculdade de Ciência da Computação da UNIP, como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação sob a orientação do professor Mestre Marcos Vinicius Gialdi.

Limeira

2017

Guilherme de Pontes

ANÁLISE E COMPARAÇÃO DE DESEMPENHO ENTRE SGBD RELACIONAL E NÃO-RELACIONAL

Trabalho de conclusão de curso apresentado à banca examinadora da Faculdade de Ciência da Computação da UNIP, como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação sob a orientação do professor Mestre Marcos Vinicius Gialdi.

Aprovado em __ de ____ de 201__.

BANCA EXAMINADORA

_____

Me. Marcos Vinicius Gialdi

_____

_____

DEDICATÓRIA

Dedico este trabalho aos meus pais, pessoas especiais que fizeram a diferença; Aos meus professores Marcos Vinicius Gialdi, Antônio Mateus Locci e Sérgio Nunes por todo apoio durante a produção do mesmo.

“Descobrir consiste em olhar para o que todo mundo está vendo e pensar uma coisa diferente”. (Roger Von Oech)

RESUMO

Com o aumento do uso de redes sociais e serviços Web em geral, houve um aumento excessivo na quantidade de informações armazenadas em um banco de dados. Devido a este aumento, há perda de desempenho ao lidar com grande quantidade de dados no modelo relacional. O modelo não relacional (NoSQL) surgiu para suprir esta necessidade, ou seja, tratar de forma eficiente as informações em massa. Com o intuito de comprovar o alto desempenho que é característico em bancos de dados não relacionais, foi desenvolvido uma ferramenta utilizando NodeJS para testar dois sistemas de gerenciamento de banco de dados, sendo eles o MySQL(modelo relacional) e o MongoDB(não relacional orientado a documentos). Com esta ferramenta é possível definir a quantidade de registros a serem manipulados através de querys para inserir, consultar e remover informações no banco de dados para que no fim da execução exiba o tempo em milissegundos ao finalizar sua execução com intuito de analisar de maneira eficaz o tempo gasto para conclusão das operações.

Palavras-chave: MongoDB, MySQL, relacional, não-relacional.

ABSTRACT

Through the increase of the use of social's networking and Web's services in general, has been an excessive increase in the amount of information stored in the databases. Owing to this increase, there's loss of performance while managing large amounts of data in the relational model. The non-relational model (NoSQL) arose to supply this need, that's, to accomplish efficiently the large amount of information. In order to verify the high performance in the non-relational's databases, a tool was developed using NodeJS to test two database management systems, named MySQL (relational model) and MongoDB (non-relational document-oriented). With this tool it's possible has set the amount of records to be manipulated through querys to insert, query and remove information in the database and at the end of the execution, so displays the time in milliseconds to analyze the time to complete operations.

KeyWords: MongoDB, MySQL, relational, no-relational

LISTA DE ABREVIATURAS

NoSQL	Not Only SQL
SGBD	Sistema de Gerenciamento de Banco de Dados
JSON	JavaScript Object Notation
BSON	Binary JSON
XML	Extensible Markup Language
SQL	Structured Query Language
DML	Data Manipulation Language
DDL	Data Definition Language
ASCII	American Standard Code for Information Interchange
TI	Tecnologia da Informação
RAM	Random Access Memory
CPU	Central Processing Unit
MVCC	Multiversion Concurrency Control
RDBMS	Relational Database Management System
ORM	Object Relational Mapping
HDFS	Hadoop Distributed File System

LISTA DE FIGURAS

Figura 1: Arquitetura simples com base no modelo relacional. 17
Figura 2: Exemplo de escalabilidade horizontal e vertical. 23
Figura 3: Exemplo de MapReduce. 27
Figura 4: MongoDB sem a utilização de shard. 28
Figura 5: MongoDB com a utilização de shard. 28
Figura 6: Escalonamento das informações ao utilizar sharding. 29
Figura 7: Querys básicas MySQL. 34
Figura 8: Querys básicas MongoDB. 34
Figura 9: Teorema CAP. 37
Figura 10: Máquina virtual executando simultaneamente duas bases de dados. 45
Figura 11: Diagrama de entidade e relacionamento do banco de dados relacional. 45
Figura 12: Documento JSON. 46
Figura 13: Código fonte do index.js 48
Figura 14: Relacionamento das tabelas Player e Team(MySQL). 50
Figura 15: Gráfico em função do teste de inserção. 51
Figura 16: Consumo de disco excessivo ao inserir informações em massa no MySQL. 52
Figura 17: Armazenamento de memória física – MongoDB. 52
Figura 18: Armazenamento de memória física – MySQL. 53
Figura 19: Gráfico em função do teste de busca. 53
Figura 20: Gráfico em função do teste de exclusão 54

LISTA DE QUADROS

Quadro 1: Principais vantagens do modelo relacional. 18
Quadro 2: Principais desvantagens do modelo relacional. 19
Quadro 3: Vantagens e desvantagens do modelo orientado a colunas. 30
Quadro 4: Vantagens e desvantagens do modelo chave-valor. 31
Quadro 5: Vantagens e desvantagens do modelo orientado a documentos. 32
Quadro 6: Vantagens e desvantagens do modelo orientado a grafos. 32
Quadro 7: Terminologias SQL RDBMS e MongoDB. 33
Quadro 8: Software e hardware da máquina virtual. 44
Quadro 9: Versão dos SGBD utilizados. 44

SUMÁRIO

1 INTRODUÇÃO 13

2 MODELO RELACIONAL 15

2.1 História 15
2.2 Arquitetura 16
2.3 Vantagens 18
2.4 Desvantagens 19

3 NOSQL 21

3.1 História 21
3.2 Big Data 22
3.3 Características 22
3.3.1 Escalabilidade Horizontal 22
3.3.2 Schema-Free 24
3.3.3 MVCC(Multiversion concurrency control) 24
3.3.4 Vector clocks 26
3.3.5 Map Reduce 26
3.3.6 Sharding 27
3.4 Cenários 29
3.5 Modelos 30
3.5.1 Orientado a colunas 30
3.5.2 Chave-valor 31
3.5.3 Orientado a documentos 32
3.5.4 Orientado a grafos 32

4 CARACTERÍSTICAS E COMPARAÇÕES 33

4.1 Terminologia 33
4.2 Sintaxe 33
4.3 ACID X BASE 34
4.3.1 ACID 34
4.3.2 BASE 35
4.4 Teorema CAP 36

5 IMPORTÂNCIA 39

6 TECNOLOGIAS 41

6.1 MySQL 41
6.2 MongoDB 41
6.3 NodeJS 42

7 METODOLOGIA DO TESTE 43

7.1 Ambiente de Testes 43
7.2 Arquitetura utilizada 43
7.3 Máquina Virtual 43
7.3.1 Configurações 44
7.4 Banco de Dados 44
7.5 MySQL 45
7.5.1 Diagrama de Entidade e Relacionamentos 45

7.6 MongoDB 46
7.6.1 Documento JSON 46
7.7 Procedimento do Teste 47
7.8 Série de Teste 47
7.9 Programa para Teste 47
7.10 Método de realização do Teste 48
7.11 Comportamento do Teste 50
8 CONCLUSÃO E TRABALHOS FUTUROS 55
9 REFERÊNCIAS 56
10 APÊNDICES 57

1. INTRODUÇÃO

Com a popularização da internet é gerado uma alta quantidade de informações para serem armazenadas em um banco de dados, tornando necessário a implementação de conceitos e práticas de *Big Data*. O modelo relacional de banco de dados tende a perder desempenho ao manipular um extenso volume de informações, surgindo assim, um movimento que atende esta necessidade, disponibilizando alta performance ao gerenciar dados em massa, movimento este denominado por NoSQL(*Not only SQL*). É importante ressaltar que o movimento NoSQL não tem como intuito substituir o modelo relacional, mas sim, permitir um melhor desempenho na utilização de banco de dados, que muita das vezes não está presente em bancos relacionais.

O *Big Data* não é o único motivo para a ascensão do paradigma NoSQL. A facilidade para se utilizar *clusters* e também simplificar a comunicação entre aplicações são fatores cruciais para a implementação desta tecnologia. (SADALAGE; FOWLER, 2013)

Este volume de informações está geralmente ligado a Redes Sociais e aplicações Web, pois, há uma interação constante por parte dos usuários que utilizam este tipo de serviço. Possuem uma estrutura de armazenamento que expande seu tamanho, sua complexidade de interação e assim causa a lentidão no mapeamento dos registros quando utilizado o modelo relacional. No entanto, atualmente se faz uso de modelos não-relacionais, o que melhora a performance e escalabilidade. (WIESE, 2015)

O NoSQL(*Not only SQL*), é um paradigma não-relacional para armazenamento de informações, pois a ideia é auxiliar e aperfeiçoar o escalonamento horizontal dos dados, ou seja, sua implementação será flexível em ambientes com *cluster*, são bem aprimorados e simples em relação ao modelo relacional. Outra vantagem do modelo não-relacional é possuir um alto desempenho na obtenção de uma grande quantidade de dados, e este detalhe é o que motivou diversas empresas de grande porte a aderirem essa nova tecnologia. (SADALAGE; FOWLER, 2013)

A primeira versão do MongoDB, foi publicada em 2009 e surgiu como uma solução emergente. O banco de dados em questão é uma variação do paradigma NoSQL e suas principais características baseiam-se em ser orientado a documentos, livre de

esquemas e possuir alto desempenho. Uma das razões pela qual esta tecnologia foi desenvolvida, foi para suprir a necessidade de ser escalonado com facilidade. A ideia geral de um banco de dados orientado a documentos, é substituir as “Linhas” por “Documentos”, o que torna o SGBD mais flexível. (HOWS et al., 2015)

Normalmente, isso se adequada muito bem ao modo que o desenvolvedor pensa, ao utilizar programação orientada a objetos. Ao contrário do modelo relacional, o MongoDB é livre de *schemas*, pois as chaves de um determinado documento não são fixas ou pré-definidas e as migrações de dados tornam-se dispensáveis. Com a possibilidade de ser utilizado novas chaves, acaba sendo desnecessário exigir que todos os dados tenham a mesma forma. Esta melhoria dá aos programadores uma alta flexibilidade na forma como é trabalhado os modelos de dados evolutivos. (HOWS et al., 2015)

2. MODELO RELACIONAL

Dentre todas as funções de um computador, uma das mais importantes é armazenar e gerenciar informações. Uma maneira versátil de organizar informações é armazená-las em tabelas. O modelo relacional está centrado nessa ideia: a organização de dados em coleções de tabelas bidimensionais chamadas "relações".

2.1 HISTÓRIA

Em 1970, Edgar F. Codd, matemático educado em Oxford que trabalhou na empresa IBM San Jose Research Lab, publicou um artigo que mostra como as informações armazenadas em grandes bancos de dados podem ser acessadas sem saber como o dado foi estruturado ou onde residiu no banco de dados. (CODD, 1990)

Até então, a recuperação de informações exigia conhecimentos informáticos relativamente sofisticados, ou até mesmo os serviços de especialistas que sabiam desenvolver programas para buscar informações específicas - uma tarefa demorada e dispendiosa. (CODD, 1990)

O que o Codd fez foi abrir a porta para um novo mundo de independência de dados. Os usuários não precisariam ser especialistas, nem precisavam saber onde estava armazenadas as informações ou como o computador as recuperou. Eles agora poderiam se concentrar mais em seus negócios e menos em seus computadores. (CODD, 1990)

Codd chamou seu artigo, "Um Modelo Relacional de Dados para Grandes Bancos de Dados Compartilhados". Os cientistas da computação chamaram isso de "ideia revolucionária".

Hoje, a facilidade e a flexibilidade das bases de dados relacionais tornaram-se a escolha predominante para registros financeiros, fabricação e informações logísticas e dados pessoais. A maioria das transações de dados de rotina - acessar contas bancárias, usar cartões de crédito, negociar ações, fazer reservas de viagens, realizar compras *online* - todas as estruturas de uso são baseadas na teoria do banco de dados relacional. (DATE, 2001)

Codd, que havia adicionado um doutorado em ciência da computação para o seu fundo de matemática, quando ele decidiu ir para os Estados Unidos da sua Inglaterra natal, se dispôs a resolver este problema. Ele começou com uma premissa elegantemente simples: queria poder pedir ao computador informações e, em seguida, deixar o computador descobrir onde e como as informações são armazenadas e como recuperá-la. (CODD, 1990)

Em seu artigo, Codd escreveu que usava o termo relação no sentido matemático da teoria dos conjuntos, como na relação entre grupos de conjuntos. Em termos simples, sua solução de banco de dados relacional forneceu um nível de independência de dados que permitiu aos usuários acessar informações sem ter que mostrar detalhes da estrutura física de um banco de dados. (DATE, 2001)

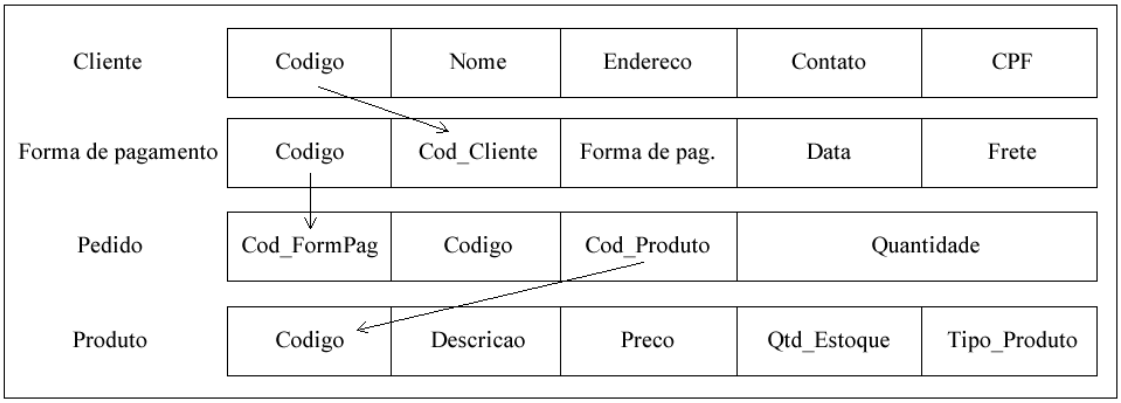
Don Chamberlin e Ray Boyce inventaram o SQL(*Structured Query Language*), hoje o idioma de computador mais utilizado para consultar bancos de dados relacionais. (CHAMBERLIN, 1996)

2.2 ARQUITETURA

O modelo mais popular de design de banco de dados é o modelo de banco de dados relacional. Este modelo foi desenvolvido para superar os problemas de complexidade e inflexibilidade dos dois modelos, Hierárquico e modelo em Rede, no gerenciamento de bases de dados com relacionamentos entre entidades diversas. (TAYLOR, 2016)

Este modelo não é apenas simples, mas também poderoso. No banco de dados relacional, cada arquivo é percebido como um arquivo plano (uma tabela bidimensional) consistindo em muitas linhas (registros), cada registro com itens de dados chave e não-chave. Os elementos da chave são os elementos de dados que identificam a gravação, ou seja, as chaves funcionam como um identificador para o registro em específico. A figura abaixo mostra os arquivos e os campos que cada registro deve ter em um sistema de faturamento do cliente. (TAYLOR, 2016)

Figura 1: Arquitetura simples com base no modelo relacional.



Fonte: O Autor.

Nesses arquivos, os itens de dados principais são identificação do cliente, número da nota fiscal e código do produto. Cada um dos arquivos podem ser usados separadamente para gerar relatórios. No entanto, os dados também podem ser obtidos a partir de qualquer combinação de arquivos, pois todos esses estão relacionados entre si com a ajuda de itens de dados chave especificados acima.

Esta é a vantagem fundamental do modelo relacional de banco de dados, além de sua simplicidade e robustez.

O modelo relacional desenha muito o trabalho de E.F. Codd, que identifica os recursos de uma boa base de dados relacional.

Todas as informações são logicamente representadas como tabelas e o acesso de dados é possível pelos nomes dos campos. Assim, a ligação de ordem, posição ou arquivo não é motivo de preocupação para os usuários. (DATE, 2015)

O dicionário de dados tem informações sobre a estrutura do banco de dados, incluindo o tipo de dados, tamanho, definições, relacionamentos e permissões de acesso. Os usuários autorizados podem aprender sobre o ambiente do banco de dados e alterar o mesmo usando a linguagem de descrição de dados(DDL). (TAYLOR, 2016)

Uma linguagem de manipulação de dados(DML) está disponível para usuários, incluindo programadores para criação, inserção, modificação, recuperação, organização e exclusões de qualquer parte do banco de dados. Essas manipulações são possíveis tanto

no nível recorde quanto em todo o arquivo, oferecendo flexibilidade na definição de permissões de acesso para várias categorias de usuários. (TAYLOR, 2016)

Qualquer modificação na estrutura do banco de dados em termos de dividir a tabela horizontalmente ou verticalmente não deve ter qualquer impacto na lógica do programa usando o banco de dados. Esta independência de dados é a principal vantagem do modelo relacional. (DATE, 2015)

A independência distribuída de dados é outra característica de um bom banco de dados relacional. Os programas de usuário não exigem nenhuma alteração quando os dados são distribuídos pela primeira vez ou redistribuídos. A localização física real dos dados não é importante para o usuário, desde que esse campo apareça no dicionário de dados como se estivesse local. (TAYLOR, 2016)

O modelo relacional consiste de dois identificadores: Chave primária e Chave estrangeira. Respectivamente a primeira chave é utilizada como o identificador de um determinado registro dentro de uma tabela. Já a chave estrangeira é utilizada para estabelecer um relacionamento entre duas tabelas. Desta forma, uma coluna irá corresponder à mesma coluna, que por sua vez é a chave primária de outra tabela. (TAYLOR, 2016)

2.3 VANTAGENS

Vantagens que são levadas em consideração ao se utilizar o modelo relacional em banco de dados:

Quadro 1: Principais vantagens do modelo relacional.

Facilidade de uso	A revisão de qualquer informação como tabelas consistindo de linhas e colunas é muito mais fácil de entender.
Flexibilidade	Diferentes tabelas à partir das quais as informações devem ser vinculadas e extraídas podem ser facilmente manipuladas por operadores, tais como projeto e junção para fornecer informações na forma em que é desejável.
Precisão	O uso de álgebra relacional e cálculo relacional na manipulação das relações entre as tabelas garante que

	não há ambiguidade, que de outra forma pode surgir ao estabelecer os vínculos em um banco de dados de tipo de rede complicado.
Segurança	O controle e autorização de segurança também podem ser implementados com mais facilidade, movendo atributos sensíveis em uma determinada tabela para uma relação separada com seus próprios controles de autorização. Se o requisito de autorização o permitir, um atributo específico pode ser incorporado novamente com outros para permitir a recuperação completa da informação.
Independência dos dados	A independência dos dados é alcançada com mais facilidade com a estrutura de normalização usada em um banco de dados relacional do que na árvore mais complicada ou na estrutura da rede.
Linguagem de Manipulação de Dados	A possibilidade de responder a consulta por meio de uma linguagem baseada em álgebra relacional e cálculo relacional, por exemplo, SQL é fácil na abordagem do banco de dados relacional. Para os dados organizados em outra estrutura, a linguagem de consulta torna-se complexa ou extremamente limitada em suas capacidades.

Fonte: Adaptado, Taylor, p. 5, 2016.

2.4 DESVANTAGENS

Desvantagens a serem consideradas e analisadas antes de se utilizar um modelo relacional para banco de dados:

Quadro 2: Principais desvantagens do modelo relacional.

Desempenho	Uma grande restrição e, portanto, uma desvantagem no uso do sistema de banco de dados relacional é o desempenho da máquina. Se o número de tabelas entre quais relacionamentos a serem estabelecidos é grande, as próprias tabelas afetam o desempenho em responder às consultas SQL.
Consumo de armazenamento físico	Com um sistema interativo, por exemplo, uma operação como associação dependeria também do armazenamento físico. É, portanto, comum em bancos de dados relacionais para sintonizar os bancos de dados e, nesse caso, o layout de dados físicos será escolhido

	de modo a dar um bom desempenho nas operações mais frequentemente executadas. Isso, naturalmente, resultaria no fato de que as operações comumente executadas tendem a se tornar ainda mais compartilhadas.
Extração lenta de significado a partir de dados	Se os dados são naturalmente organizados de forma hierárquica e armazenados como tais, a abordagem hierárquica pode dar um significado rápido para esses dados.

Fonte: Adaptado, Taylor, p. 5, 2016.

3. NOSQL

3.1 HISTÓRIA

Pela primeira vez, o termo NoSQL foi usado no final dos anos 90. O significado real do paradigma utilizado atualmente, só ocorreu em meados de 2009. Originalmente, era um título do banco de dados de código aberto criado por Carlo Strozzi, que armazena todos os dados como arquivos ASCII e *shell script*, em vez de SQL para acessar dados. Este banco de dados não tinha nada em comum com o NoSQL na forma atual. (SADALAGE; FOWLER, 2013)

O termo NoSQL tem uma origem absolutamente natural e não possui uma definição ou instituição universalmente aceita por trás. Este título é bastante caracterizado pelo vetor de desenvolvimento de TI, distante de bancos de dados relacionais. Pramod J. Sadalage e Martin Fowler tentaram agrupar e organizar o conhecimento sobre o mundo NoSQL no livro "NoSQL Essencial". (SADALAGE; FOWLER, 2013)

O paradigma NoSQL está ganhando popularidade com enorme velocidade. No entanto, isso não significa que bancos de dados relacionais estão se tornando algo arcaico. Vivemos em uma era de persistência poliglota, uma era de diferentes necessidades em um *data warehouse* diferenciado. O modelo relacional não deve ser tratado como o modelo padrão ao se desenvolver uma aplicação, sabendo-se que atualmente existem inúmeras formas de se armazenar dados. Ao tratar o modelo relacional como padrão, dependendo da quantidade de informações armazenadas no banco de dados, o desenvolvedor estará penalizando o usuário final. (SADALAGE; FOWLER, 2013)

Este movimento não tem o propósito de substituir o modelo relacional, mas sim, resolver problemas apresentados em bancos de dados relacionais, como por exemplo aumentar o desempenho na busca e persistência de grande quantidade de informações. O paradigma tem características únicas, dentre elas a escalabilidade horizontal dos dados, que possibilita o armazenamento em locais diferentes, utilizando vários nós. Outro exemplo seria realizar o armazenamento de um banco de dados em *cloud*, atingindo um alto nível de paralelismo. (SADALAGE; FOWLER, 2013)

3.2 BIG DATA

Big Data é um termo usado para dar ênfase a um volume excessivo de dados estruturados e não-estruturados que por ser tão grande, é difícil de processar usando técnicas tradicionais de banco de dados. Na maioria dos cenários empresariais, o volume de dados é muito grande, se move muito rápido ou excede a capacidade de processamento atual. (DUMBILL, 2012)

A Big Data tem potencial para ajudar as empresas a melhorar as operações e tomar decisões mais rápidas e inteligentes. Esses dados, quando capturados, formatados, manipulados, armazenados e analisados, podem ajudar uma empresa a obter informações úteis para aumentar as receitas, obter ou reter clientes e melhorar as operações. (DUMBILL, 2012)

Embora o termo aparenta referenciar o volume de dados, isso nem sempre é o caso. O termo *Big Data*, especialmente quando usado pelos vendedores, pode se referir à tecnologia (que inclui ferramentas e processos) que uma organização precisa para lidar com grandes quantidades de dados e instalações de armazenamento. Acredita-se que o termo se tenha originado com as empresas de pesquisa da *Web* que precisavam consultar agregações distribuídas muito grandes de dados estruturados sem restrições. (DUMBILL, 2012)

3.3 CARACTERÍSTICAS

O modelo não-relacional consiste de características e funcionalidades, que tornam este, tão diferente do modelo relacional.

3.3.1 ESCALABILIDADE HORIZONTAL

Existem dois tipos de escalabilidade, sendo elas vertical e horizontal. A escalabilidade vertical consiste em melhorar a capacidade de processamento e armazenamento das máquinas. Já a escalabilidade horizontal resume-se em aumentar a quantidade de nós disponíveis. A principal característica do modelo não-relacional está

na facilidade de implementação da escalabilidade horizontal, pois, em comparação com a escalabilidade vertical é a mais viável. Esta tecnologia de alta disponibilidade é adequada para a solução dos problemas no gerenciamento de grande volume de dados. Existem inúmeras técnicas para que se alcance a alta disponibilidade, uma delas, é o *Sharding*. (FOWLER, 2015)

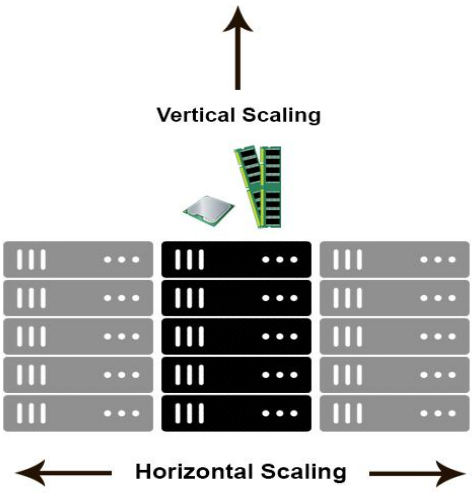
Em banco de dados, a escalabilidade horizontal geralmente é baseada no particionamento dos dados, ou seja, cada nó contém apenas uma parte dos dados. Já na escalabilidade vertical, os dados residem em um único nó e a escala é feita através de múltiplos núcleos, ou seja, espalhando a carga entre os recursos de CPU e Memória RAM dessa máquina. (SADALAGE; FOWLER, 2013)

O dimensionamento vertical geralmente é limitado à capacidade de uma única máquina, a escala além dessa capacidade geralmente envolve tempo de inatividade e vem com um limite superior. (FOWLER, 2015)

Dois exemplos de SGBD que utilizam escalabilidade horizontal: Cassandra e MongoDB. Um outro exemplo para escalabilidade vertical é o MySQL. Ele fornece uma maneira fácil de escalar verticalmente, mudando de máquinas pequenas para maiores. Esse processo geralmente envolve tempo de inatividade.

É possível ilustrar a escalabilidade horizontal e vertical da seguinte forma:

Figura 2: Exemplo de escalabilidade horizontal e vertical.



Fonte: Stack Overflow, 2016.

3.3.2 SCHEMA-FREE

A ausência de esquema (*schema-free*) é uma característica forte dos bancos de dados NoSQL, é essa falta de esquema que permite de maneira fácil a escalabilidade e aumento da disponibilidade. Mas devido a isso, bancos de dados NoSQL não garantem a integridade dos dados, diferente dos modelos relacionais que mantém uma estrutura rígida. (SADALAGE; FOWLER, 2013)

Um banco de dados sem esquema é um banco de dados que armazena suas informações sem uma estrutura anterior. Por exemplo, em um banco de dados de esquema tradicional, antes de começar a adicionar registros, é necessário definir a estrutura de seus dados. No MySQL por exemplo, é possível declarar colunas para uma tabela e, em seguida, adicionar linhas que tenham valores respectivos para cada uma das colunas. Este SGBD não permite que seja inserido em uma tabela, informações que não foram declaradas anteriormente, na criação da mesma, ou seja, este SGBD exige que seja criado um esquema estruturado, antes de inserir registros em uma tabela e caso contrário, a inserção será rejeitada. (SADALAGE; FOWLER, 2013)

Em bases de dados sem esquemas, como por exemplo, o MongoDB, é possível adicionar registros sem qualquer estrutura anterior. Além disso, pode-se agrupar registros que não possuem a mesma estrutura, ou seja, é possível ter uma coleção (funciona como uma tabela em bancos de dados relacionais onde pode-se agrupar informações) com registros de várias estruturas, em outras palavras, os registros não precisam ter as mesmas colunas(propriedades). (HOWS et al., 2015)

3.3.3 MVCC(*Multiversion concurrency control*)

O MVCC é um mecanismo que suporta transações paralelas em um determinado banco de dados. Permite que operações de escrita e leitura sejam realizadas simultaneamente, diferentemente do esquema padrão de gerenciamento de transações, pois não faz uso de locks. (FOWLER, 2015)

O MVCC é um método de controle de concorrência comumente usado por sistemas de gerenciamento de banco de dados para fornecer acesso simultâneo ao banco

de dados e em linguagens de programação para implementar a memória transacional. (FOWLER, 2015)

Caso uma operação de leitura esteja sendo realizada simultaneamente com uma operação de escrita, é possível que o leitor veja um registro de dado meio escrito ou inconsistente. Existem várias maneiras de resolver esse problema, conhecidos como métodos de controle de concorrência. A maneira mais simples é fazer com que todos os leitores esperem até que o gravador esteja pronto, o que é conhecido como um bloqueio. Isso pode ser muito lento, então o MVCC usa uma abordagem diferente: cada usuário conectado ao banco de dados vê um registro instantâneo do banco de dados em um instante particular. Qualquer alteração feita por um escritor não será vista por outros usuários do banco de dados até que as alterações tenham sido concluídas (ou, em termos de banco de dados: até que a transação tenha se aplicado o *commit*). (FOWLER, 2015)

Quando um banco de dados MVCC precisa atualizar um item do registro, ele não substituirá os dados antigos com novos dados, mas marca o registro antigo como obsoleto e adiciona a versão mais recente em outro lugar. Assim, existem várias versões armazenadas, mas apenas uma é a mais recente. Isso permite que os leitores acessem os dados que estavam lá quando começaram a ler, mesmo que fossem modificados ou excluídos parcialmente por outra pessoa. Também permite que o banco de dados evite a sobrecarga de preenchimento de furos na memória ou estruturas de disco, mas requer que o sistema geralmente pare de varrer periodicamente e excluir os objetos de dados velhos e obsoletos. Para um banco de dados orientado a documentos, ele também permite que o sistema otimize documentos escrevendo documentos inteiros em seções contíguas de disco - quando atualizado, o documento inteiro pode ser reescrito ao invés de bits e registros cortados ou mantidos em um link. (FOWLER, 2015)

O MVCC fornece pontos de vista consistentes. As transações de leitura sob o MVCC normalmente utilizam um registro único de data/hora ou identificação de transação para determinar o estado do banco de dados a ser lido. As transações de leitura e escrita são assim isoladas uma da outra sem necessidade de bloqueio. (FOWLER, 2015)

3.3.4 VECTOR CLOCKS

Por conta das transações serem executadas simultaneamente, é necessário que seja armazenado um arquivo de *log*, indicando qual versão de um dado distribuído é o mais recente. (SADALAGE; FOWLER, 2013)

Alguns bancos de dados, como por exemplo o Cassandra, utilizam timestamps para determinar qual é a transação mais recente. Se houver duas modificações conflitantes em um valor de coluna, aquele com o timestamp mais alto será considerado o mais atual, sendo assim este será o registro correto. (SADALAGE; FOWLER, 2013)

O relógio vetorial tem a vantagem de não exigir sincronização de hora em todos os nós, e auxilia na identificação das transações que podem estar em conflito. (SADALAGE; FOWLER, 2013)

3.3.5 MAP REDUCE

Outro paradigma utilizado pelas tecnologias não-relacionais é o *MapReduce*. Este termo tem o objetivo de separar os processos em duas etapas:

- Map: Mapeia e divide os dados em diversos nós de armazenamento e processamento.
- Reduce: Junta e processa os resultados fragmentados para gerar um resultado final.

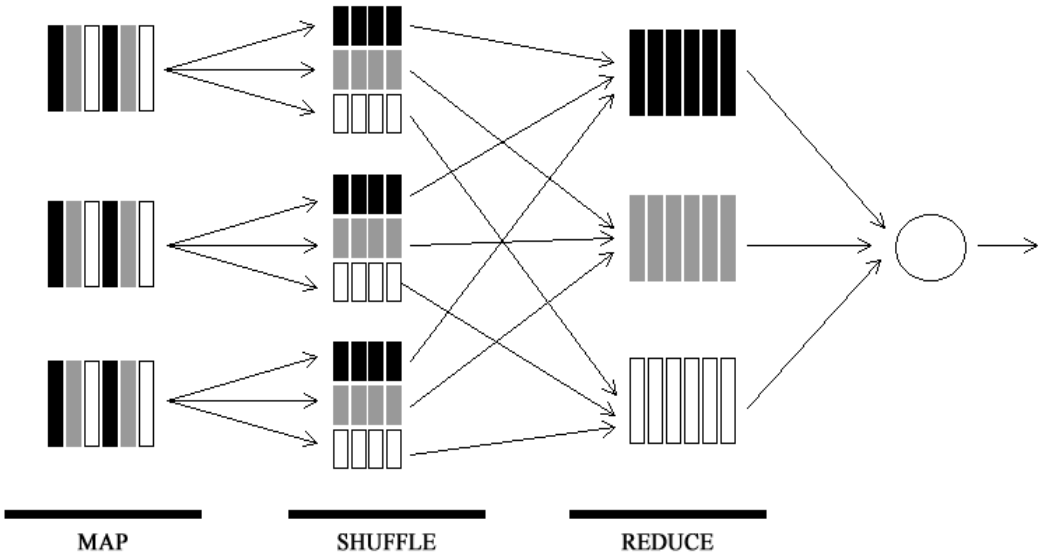
É este paradigma de programação que permite a escalabilidade maciça em centenas ou milhares de servidores em um cluster. O conceito *MapReduce* está ligado a soluções de processamento de dados em escala de cluster. (SADALAGE; FOWLER, 2013)

O termo *MapReduce* refere-se a duas tarefas distintas. O primeiro é o trabalho do mapa, que leva um conjunto de dados e o converte em outro conjunto de dados, onde os elementos individuais são divididos em tuplas (pares chave/valor). O trabalho de redução leva o resultado de um mapa como entrada e combina essas tuplas de dados em um conjunto menor de tuplas. Como a sequência do nome *MapReduce* implica, o trabalho de

redução sempre é realizado após o trabalho de mapeamento. (SADALAGE; FOWLER, 2013)

Para ilustrar como é realizado o processo de *MapReduce* de um problema, segue abaixo uma figura que representa tal processo:

Figura 3: Exemplo de MapReduce.



Fonte: Adaptado, SADALAGE; FOWLER, p. 107, 2013.

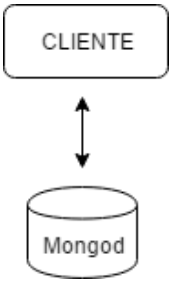
3.3.6 SHARDING

O conceito de *sharding* consiste basicamente em quebrar as coleções em pedaços menores com o intuito de serem distribuídos através de shards (fragmentos), de modo que cada *shard* seja responsável por um subconjunto do conjunto total de dados. O *software* não tem a necessidade de saber como essa fragmentação é implementada. Esse processo de roteamento é denominado por Mongos. O processo em questão sabe onde todos os dados estão localizados e a aplicação fica conectada a um processo Mongos que faz o roteamento das informações. O Mongos ao identificar como os dados foram configurados e fragmentados, envia as solicitações para o *shard* apropriado. (HOWS et al., 2015)

Na configuração de um banco de dados MongoDB que não utiliza *shard*, haveria no caso um cliente ligado a um processo Mongod (*shard*). Já em uma configuração com *shard*, o cliente se conecta a um processo Mongos, que ignora o *sharding* do *software*.

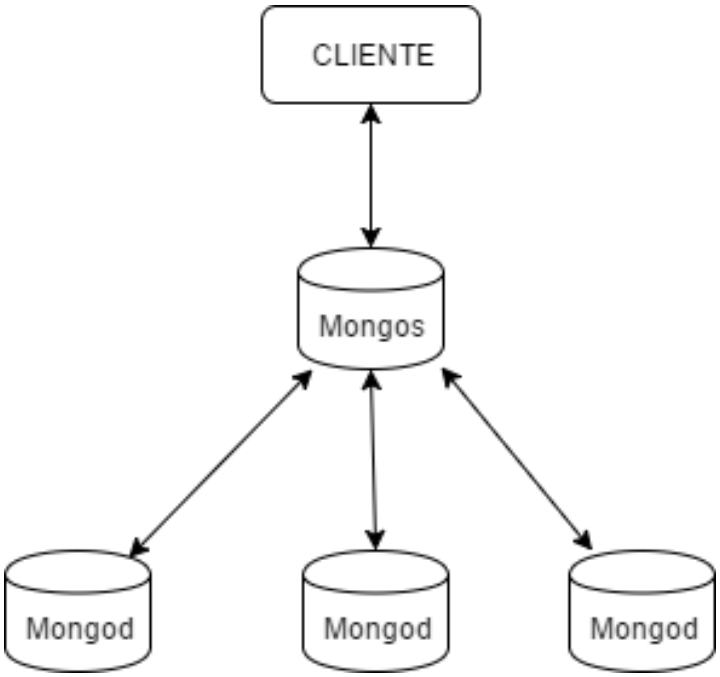
Do ponto de vista do *software*, a configuração com *shard* é idêntica a uma configuração sem *shard*, sendo assim não há necessidade de alterar o código fonte do *software*, quando houver necessidade de escalar. (HOWS et al., 2015)

Figura 4: MongoDB sem a utilização de shard.



Fonte: MongoDB Documentation, 2017.

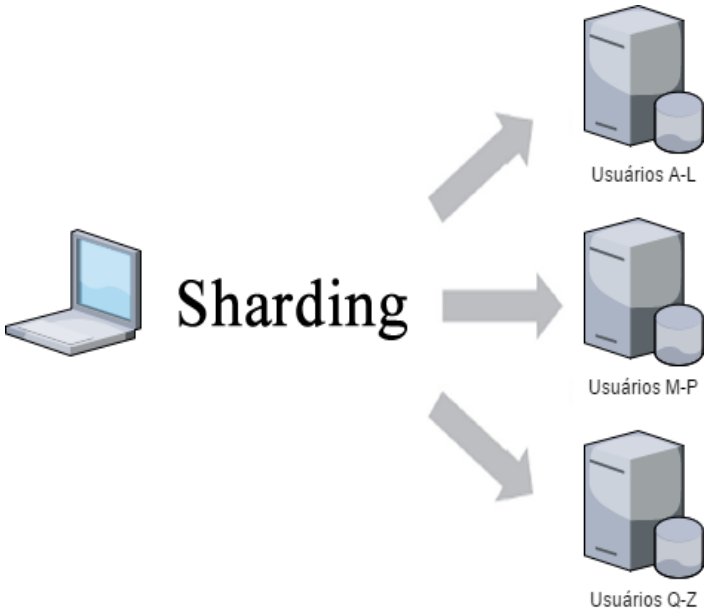
Figura 5: MongoDB com a utilização de shard.



Fonte: MongoDB Documentation, 2017.

Com a intenção de esclarecer a função *Sharding*, abaixo segue um outro exemplo de escalonamento:

Figura 6: Exemplo de como é realizado o escalonamento das informações ao utilizar sharding.



Fonte: O Autor.

Supõe se que nesse esquema, as partições são divididas entre os servidores e tal divisão é baseada no nome dos usuários por ordem alfabética de cada documento. Esta seria uma solução simplificada de como é realizado o *sharding*.

3.4 CENÁRIOS

Esta seção tem como intuito apresentar cenários em que o MongoDB se sobressai e em outros casos acaba não sendo útil.

De antemão será apresentado onde não deve ser utilizado o MongoDB. Se houver necessidade de desenvolver um sistema, que realiza inúmeras ações atomicamente, como por exemplo, uma conta bancária em que o dinheiro precisa ser creditado em uma conta e de forma automática debitado em outra, o banco de dados MongoDB terá dificuldades em lidar com esta situação. Este banco de dados não aceita ações atômicas em mais de uma coleção simultaneamente. Esta desvantagem também é válida se houver a

necessidade de desenvolver uma loja virtual que precisa realizar o gerenciamento do estoque que está em outra coleção de maneira atômica, no caso de um não pode ser feito sem o outro. Neste tipo de cenário o ideal é utilizar o modelo relacional. De fato o MongoDB não aceita operações atômicas para múltiplas coleções ao mesmo tempo. Apenas quando uma coleção for alterada e ocorrer uma atualização, inserção ou exclusão de quaisquer documentos no MongoDB, este processo sim será atômico. (HOWS et al., 2015)

Por outro lado, se a finalidade do *software* é distinta da apresentada, preferencialmente se o projeto é voltado para um *software* social – designado para suportar a interação de inúmeras pessoas – como por exemplo o Twitter, Facebook e Instagram, o MongoDB terá uma ótima performance. Este SGBD foi desenvolvido para suportar objetos multiestruturados, sendo assim, se a aplicação possui uma página, e esta possui seguidores, comentários ou diversas pessoas associadas, certamente a solução será melhor projetada. Esta solução também é válida para *softwares desktop* que não exigem atomicidade entre transações. (HOWS et al., 2015)

3.5 MODELOS

3.5.1 ORIENTADO A COLUNAS

Um SGBD orientado a coluna é um sistema de gerenciamento de banco de dados que armazena tabelas de dados como seções de colunas de dados e não como linhas de dados. Fisicamente as tabelas são uma coleção de colunas, cada uma delas é essencialmente uma tabela com um único campo. Esses bancos de dados geralmente são usados para sistemas analíticos, *business intelligence* e armazenamento de dados analíticos. (SADALAGE; FOWLER, 2013)

Quadro 3: Vantagens e desvantagens do modelo orientado a colunas.

Vantagem	• É possível comprimir dados significativamente, porque em uma única coluna da tabela, os dados geralmente são do mesmo tipo; • Permite um *hardware* barato e de baixa potência para aumentar a velocidade do desempenho da consulta nas 5, 10 e às vezes até 100 vezes, portanto, devido à compressão, os dados na unidade terão 5-10 vezes menos espaço do que no caso do RDBMS tradicional.

Desvantagem	• Em geral, não há transações; • Tem uma série de limitações para o desenvolvedor que tradicionalmente utiliza RDBMS.
SGBD	• Cassandra • HBase

Fonte: Adaptado, SAGALAGE; FOWLER, p.147, 2013.

3.5.2 CHAVE-VALOR

Este modelo de banco de dados não-relacional armazena as informações, similares a uma tabela hash, ou seja, os dados ou itens particulares ficam atrelados a uma chave única. (SADALAGE; FOWLER, 2013)

Quadro 4: Vantagens e desvantagens do modelo chave-valor.

Vantagem	• Não há necessidade de consultas SQL, índices, disparadores, procedimentos armazenados, tabelas temporárias, formulários, visualizações, etc; • O banco de dados chave/valor é facilmente escalável e de alto desempenho devido à sua leveza.
Desvantagem	• Limitações de bancos de dados relacionais garantem a integridade dos dados no nível mais baixo. No armazenamento chave/valor, não existe tal restrição. A integridade de dados é controlada por aplicativos. Neste caso, a mesma pode ser comprometida devido a erros no código do aplicativo; • Em um RDBMS se o modelo estiver bem desenhado, o banco de dados terá uma estrutura lógica que reflete completamente a estrutura dos dados armazenados e pode diferir da estrutura do aplicativo(os dados são independentes do aplicativo). Para um armazenamento de chave/valor, é mais difícil obter este tipo de estrutura.
SGBD	• Redis • MemcacheDB

Fonte: Adaptado, SAGALAGE; FOWLER, p.123, 2013.

3.5.3 ORIENTADO A DOCUMENTOS

Os programas são projetados para armazenar, pesquisar e gerenciar informações orientadas a documentos(dados semi-estruturados). O conceito central é um documento. A implementação do banco de dados específico orientado a documentos é diferente, mas, em geral, eles sugerem o encapsulamento e criptografia dos dados(documentos) em vários formatos padrão: XML, JSON, BSON, etc. (SADALAGE; FOWLER, 2013)

Quadro 5: Vantagens e desvantagens do modelo orientado a documentos.

Vantagem	• Idioma suficientemente flexível para consulta; • Facilidade para escalar horizontalmente.
Desvantagem	• A atomicidade na maioria dos casos é condicional.
SGBD	• MongoDB • CouchDB

Fonte: Adaptado, SAGALAGE; FOWLER, p.133, 2013.

3.5.4 ORIENTADO A GRAFOS

É um banco de dados que usa estruturas de grafos com nós, bordas e propriedades para representar e armazenar dados. Por definição, um banco de dados orientado a grafos é um sistema de armazenamento que fornece adjacência livre de índice. Isso significa que cada elemento contém um ponteiro direto para o elemento adjacente e as pesquisas de índice não são necessárias. (SADALAGE; FOWLER, 2013)

Quadro 6: Vantagens e desvantagens do modelo orientado a grafos.

Vantagem	• Muita das vezes mais rápido para conjuntos de dados associativos; • Pode escalar mais naturalmente para grandes conjuntos de dados, pois normalmente não requer operações de “JOIN”.
Desvantagem	• RDBMS pode ser usado em casos mais gerais. Os bancos de dados grafos são adequados para dados semelhantes a grafos.
SGBD	• OrientDB • Neo4j

Fonte: Adaptado, SAGALAGE; FOWLER, p.161, 2013.

4. CARACTERÍSTICAS E COMPARAÇÕES

4.1 TERMINOLOGIA

A terminologia do MongoDB é parecida com a dos bancos de dados relacionais, porém, possui algumas particularidades, como por exemplo:

Quadro 7: Terminologias SQL RDBMS e MongoDB.

SQL RDBMS	MongoDB
DATABASE	DATABASE
TABLE	COLLECTION
ROWS	DOCUMENT JSON
INDEX	INDEX
PARTITION	SHARD
QUERY	QUERY
COLUMN	FIELD
JOIN	EMBEDDED DOCUMENTS

Fonte: MongoDB Documentation, 2017.

4.2 SINTAXE

Outra particularidade, está na forma que é escrito as *querys* do banco de dados relacional MySQL em comparação ao MongoDB:

MySQL

Figura 7: Querys básicas MySQL.

```sql
/*INSERT*/
INSERT INTO player (team, name, score, email) VALUES (1, “Guilherme”, 742, “teste@teste.com.br”);

/*SELECT*/
SELECT * FROM player;

/*DROP, DELETE, */
DROP TABLE player;
DELETE FROM player WHERE nome = 'Guilherme'

/*UPDATE*/
UPDATE player SET nome = 'Guilherme de Pontes', email = 'teste@teste.com.br' WHERE id = 1;
```

Fonte: MySQL Documentation, 2017.

MongoDB

Figura 8: Querys básicas MongoDB.

```sql
/*INSERT*/
db.player.insert({'name':'Guilherme','email':'teste@teste.com.br','score':742,
'team':[{'name:':'team1','country':'Brasil','city':'Sao Paulo'}]})

/*SELECT*/
db.player.find()

/*REMOVE*/
db.player.remove()

/*UPDATE*/
db.player.update(
  { "name" : "Guilherme de Pontes" },
  { $set: { "idade": "23" } }
)
```

Fonte: MongoDB Documentation, 2017.

4.3 ACID X BASE

4.3.1 ACID

Segundo Fowler (2015), uma característica de sistemas de banco de dados relacionais é algo conhecido como conformidade ACID. O ACID é um acrônimo - letras individuais, destinadas a descrever uma característica das transações de banco de dados relacionais, podem ser expandidas conforme descrito nesta lista:

- Atomicidade: A transação do banco de dados deve ter resposta de sucesso ou falha. O sucesso parcial não é permitido.
- Consistência: Durante a transação do banco de dados, o RDBMS progride de um estado válido para outro. O estado nunca é inválido.
- Isolamento: A transação do banco de dados do cliente deve ocorrer isoladamente de outros clientes que tentam transacionar com o RDBMS.
- Durabilidade: A operação de dados que faz parte da transação deve ser refletida no armazenamento não volátil (memória do computador que pode recuperar informações armazenadas mesmo quando não é alimentado, como um disco rígido) e persistir após a conclusão da transação. As falhas de transação não podem deixar os dados em um estado parcialmente comprometido.

Ainda conforme Fowler (2015), certos casos de uso para RDBMS, como o processamento de transações *online*, dependem de transações compatíveis com ACID entre o cliente e o RDBMS para que o sistema funcione corretamente. Um ótimo exemplo de uma transação compatível com ACID é uma transferência de fundos de uma conta bancária para outra.

Isso é dividido em duas transações de banco de dados, onde a conta de origem mostra uma retirada e a conta de destino mostra um depósito. Obviamente, essas duas transações devem ser vinculadas para serem válidas para que, se uma delas falhar, toda a operação deve deixar de garantir que ambos os saldos permaneçam válidos.

4.3.2 BASE

Um conceito chave por trás dos bancos de dados NoSQL é que nem todas as aplicações realmente precisam de transações compatíveis com ACID. Desconsiderar certas propriedades do ACID (e se afastar do modelo relacional) abriu uma série de possibilidades, que permitiram que alguns bancos de dados NoSQL alcançassem escalabilidade e desempenho maciço para seus aplicativos. (FOWLER, 2015)

Segundo (FOWLER, 2015), considerando que o ACID define as principais características necessárias para o processamento confiável de transações, o mundo

NoSQL requer características diferentes para permitir flexibilidade e escalabilidade. Essas características opostas são inteligentemente capturadas na sigla BASE:

- Basicamente disponível (*Basically Available*): O sistema garante estar disponível para consulta aos usuários.
- Estado livre (*Soft State*): Os valores armazenados no sistema podem mudar devido ao eventual modelo de consistência.
- Eventualmente consistente (*Eventually Consistent*): À medida que os dados são adicionados ao sistema, o estado do mesmo é gradualmente replicado em todos os nós. Por exemplo, no Hadoop, quando um arquivo é gravado no HDFS, as réplicas dos blocos de dados são criadas em diferentes nós de dados depois que os blocos de dados originais foram escritos. Durante o curto período antes de os blocos serem replicados, o estado do sistema de arquivos não é consistente.

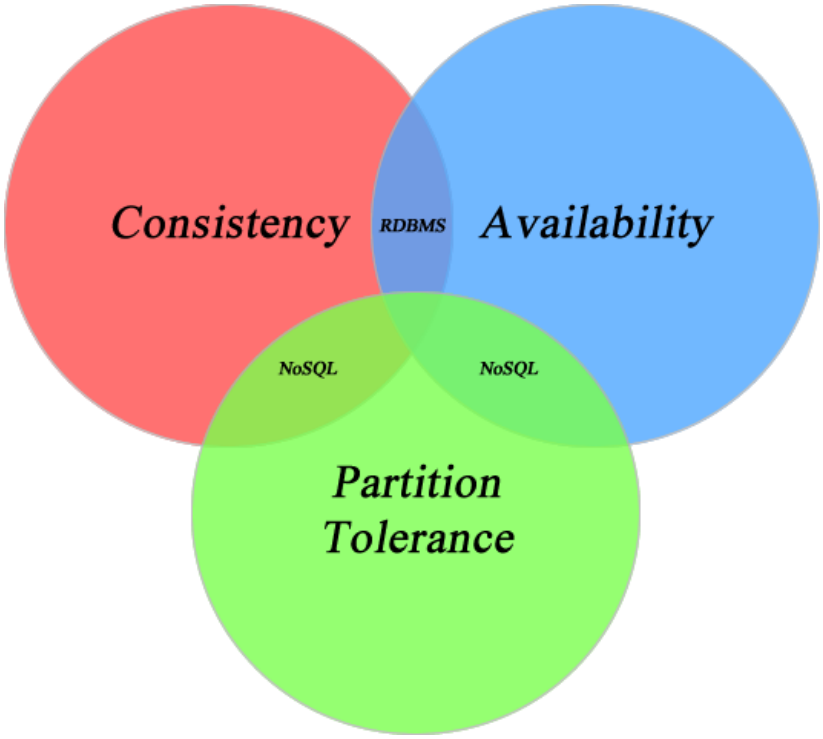
O acrônimo BASE não é totalmente independente, já que a maioria dos bancos de dados NoSQL não abandonam completamente todas as características ACID - não é realmente o conceito oposto do polo que o nome implica, em outras palavras. Além disso, as características estado livre e eventualmente consistente equivalem ao mesmo, mas o ponto é que, ao dispensar a consistência, o sistema pode escalar horizontalmente (muitos nós) e garantir a disponibilidade. (FOWLER, 2015)

4.4 TEOREMA CAP

Em 2000, Eric Brewer apresentou o seu discurso de abertura no Simpósio ACM sobre os Princípios da Computação Distribuída e o Teorema CAP nasceu. A maioria das pessoas no mundo exterior nunca ouviram falar de um teorema desse tipo, nem se importam. Eles só querem que seus PCs funcionem, a Internet funcione, as mídias sociais funcionem, todas com disponibilidade consistente em seus arquivos. O teorema do CAP, também conhecido como Teorema de Brewer, foi posteriormente revisado e alterado através do trabalho de Seth Gilbert e Nancy Lynch do MIT em 2002, além de muitos outros desde então. O princípio central do teorema afirma que existem três requisitos de sistema essenciais necessários para o projeto, implementação e implantação de

aplicativos bem sucedidos em sistemas de computação distribuída. Eles são Consistência, Disponibilidade e Tolerância de Partição ou CAP. (CARPENTER; HEWITT, 2016)

Figura 9: Teorema CAP.



Fonte: Adaptado, CARPENTER; HEWITT, p. 23, 2016.

A consistência indica se um sistema funciona completamente ou não. O sistema segue de forma confiável as regras estabelecidas dentro da sua programação de acordo com essas regras definidas. Todos os nós dentro de um cluster veem todos os dados que deveriam ser. Esta é a mesma ideia apresentada no ACID. (CARPENTER; HEWITT, 2016)

A disponibilidade significa exatamente como parece. O serviço ou sistema fornecido está disponível quando solicitado. Cada pedido obtém uma resposta de fracasso ou sucesso. (CARPENTER; HEWITT, 2016)

Tolerância de partição representa o fato de que um determinado sistema continua a funcionar mesmo em circunstâncias de perda de dados ou falhas no sistema. Uma falha de um único nó não deve fazer com que todo o sistema colapse. (CARPENTER; HEWITT, 2016)

Essas são apenas definições simples dos três aspectos do teorema CAP. Existem inúmeros documentos disponíveis que discutem as muitas interpretações, análises e questões complexas envolvidas na aplicação real do mundo real do teorema. (CARPENTER; HEWITT, 2016)

O principal motivo para apresentar o teorema é argumentar que, na maioria dos casos, um sistema distribuído só pode garantir dois dos recursos e não os três. Ignorar essa decisão pode ter resultados catastróficos que incluem a possibilidade de os três elementos caírem simultaneamente. Os constrangimentos do teorema CAP sobre a confiabilidade do banco de dados foram monumentais para novos sistemas distribuídos, não relacionais, em grande escala. Muitas vezes precisam de disponibilidade e tolerância de partição, de modo que a consistência seja dispensável. (CARPENTER; HEWITT, 2016)

5. IMPORTÂNCIA

Os testes que serão realizados neste trabalho, tem como principal objetivo, comparar o desempenho entre dois SGBD, modelo relacional e não-relacional. O teste de performance tem como intuito apresentar uma tecnologia alternativa, que deve ser levada em consideração ao realizar o levantamento de requisitos de aplicações. Normalmente o modelo relacional é adotado como padrão ao se desenvolver um *software*, e dependendo do tamanho da base de dados, que com o tempo pode ficar extremamente grande, a aplicação poderá apresentar lentidão ao buscar informações no banco de dados.

Ou seja, o usuário final não deve ser penalizado por uma falha durante o processo de análise dos requisitos, em que o banco de dados relacional foi considerado como padrão, por ser um modelo comumente utilizado.

A pesquisa por novos recursos de armazenamento de dados tem se mostrado bastante relevante, levando ao surgimento de inúmeros bancos de dados.

Esse motivo já deveria ser o bastante para incentivar os analistas a buscarem novas formas mais eficientes de armazenar e buscar os dados gerados pelo sistema em desenvolvimento. O método de comparação será utilizado para demonstrar na prática, as vantagens em se utilizar bancos de dados não-relacionais, tendo enfoque no banco de dados MongoDB.

Organizações de todos os tamanhos estão adotando o MongoDB porque permite que eles criem aplicativos mais rapidamente, manipule tipos de dados altamente diversificados e gerenciem aplicativos de forma mais eficiente em escala.

O desenvolvimento é simplificado, pois os documentos MongoDB mapeiam naturalmente para linguagens de programação modernas e orientadas a objetos. O uso do MongoDB remove a complexa camada de mapeamento objeto-relacional (ORM) que traduz objetos no código para tabelas relacionais. (HOWS et al., 2015)

O MongoDB também pode ser dimensionado dentro e entre vários centros de dados distribuídos, proporcionando novos níveis de disponibilidade e escalabilidade, anteriormente impossíveis de serem alcançados com bancos de dados relacionais como o MySQL. À medida que as suas implantações crescem em termos de volume de dados e

taxa de transferência, o MongoDB escala facilmente sem tempo de inatividade e sem alterar sua aplicação. Em contraste, alcançar escala com o MySQL requer frequentemente um trabalho de engenharia significativo e personalizado (HOWS et al., 2015). O Baidu migrou do MySQL para o MongoDB para suportar o seu negócio em rápido crescimento. O gigante chinês dos serviços de internet agora possui mais de 100 aplicativos e gerencia mais de 1PB de dados com seu cluster MongoDB. (BAIDU AND MONGODB, 2016)

6. TECNOLOGIAS

6.1 MYSQL

O MySQL é um sistema de gerenciamento de banco de dados relacional popular de código aberto (RDBMS) desenvolvido, distribuído e suportado pela Oracle Corporation. Como outros sistemas relacionais, o MySQL armazena dados em tabelas e usa linguagem de consulta estruturada (SQL) para acessar o banco de dados e manipular as informações. (WATSON, 2016)

No MySQL, é possível pré-definir o esquema de banco de dados com base nos requisitos e configurar regras para reger as relações entre os campos nas tabelas. No MySQL, as informações relacionadas podem ser armazenadas em tabelas separadas, mas associadas através do uso de JOINS. Desta forma, a duplicação dos dados é minimizada. (WATSON, 2016)

6.2 MONGODB

MongoDB é um banco de dados open-source que usa um modelo de dados orientado a documentos.

MongoDB é um dos vários tipos de banco de dados que surgiram em meados dos anos 2000 sob o paradigma NoSQL. Em vez de usar tabelas e linhas, como em bancos de dados relacionais, o MongoDB é construído em uma arquitetura de coleções e documentos. Os documentos compreendem conjuntos de pares chave-valor e são a unidade básica de dados no MongoDB. As coleções contêm conjuntos de documentos e funcionam como o equivalente a tabelas de banco de dados relacionais. (HOWS et al., 2015)

O MongoDB suporta o design do esquema dinâmico, permitindo que os documentos em uma coleção tenham diferentes campos e estruturas. O banco de dados usa um formato de armazenamento de documentos e intercâmbio de dados chamado BSON, que fornece uma representação binária de documentos similares a JSON. O corte automático permite que os dados de uma coleção sejam distribuídos em vários servidores

para assim prover escalabilidade horizontal, à medida que os volumes de dados aumentam. (HOWS et al., 2015)

Este SGBD foi criado por Dwight Merriman e Eliot Horowitz, que encontraram problemas de desenvolvimento e escalabilidade com abordagens de banco de dados relacionais tradicionais ao construir aplicativos da Web na DoubleClick, uma empresa de publicidade na Internet que agora é detida pela Google Inc. De acordo com Merriman, o nome da base de dados foi derivado da palavra Humongous para representar a ideia de suportar grandes quantidades de dados. Merriman e Horowitz ajudaram a formar a 10Gen Inc. em 2007 para comercializar o MongoDB e *softwares* relacionados. A empresa foi renomeada MongoDB Inc. em 2013. (HOWS et al., 2015)

6.3 NODEJS

O NodeJS é uma ferramenta open-source e multiplataforma para o desenvolvimento de aplicativos *server-side* e de rede. Os aplicativos do NodeJS são escritos em JavaScript e podem ser executados em sistemas operacionais GNU/Linux, OS X e Microsoft Windows. (SYED, 2014)

Esta plataforma também fornece uma biblioteca rica de vários módulos de JavaScript que simplifica o desenvolvimento de aplicativos *Web.* (SYED, 2014)

O principal motivo para utilização do NodeJS, nos testes de comparação, é devido a esta ferramenta disponibilizar inúmeros módulos que integram com flexibilidade o MongoDB e o MySQL ao código, dando liberdade ao desenvolvedor para executar *querys* de consulta, inserção, atualização e exclusão com facilidade.

Para este trabalho será utilizado a versão 0.10.29 do NodeJS, pois é uma das versões estáveis do repositório.

7. METODOLOGIA DO TESTE

7.1 AMBIENTE DE TESTES

O ambiente de testes é virtualizado, pois este tipo de ambiente possui as seguintes vantagens:

- Independência de *hardware*;
- Economia de espaço físico e energia elétrica;
- Manutenção simplificada;
- Gerenciamento centralizado;
- Em caso de desastres, é possível recuperar o ambiente com facilidade.

A única desvantagem da máquina virtual é o consumo excessivo de Memória RAM, porém, este consumo não impactou consideravelmente na prática dos testes, pois a máquina real dispõe 32Gb de memória.

7.2 ARQUITETURA UTILIZADA

Todas as tecnologias e *softwares* utilizados neste trabalho, serão executados utilizando arquitetura 64 bits.

7.3 MÁQUINA VIRTUAL

O *software* utilizado para executar a máquina virtual, será o VirtualBox 5.1.12 da empresa Oracle. Para este trabalho, será utilizado este *software* pois o mesmo é gratuito.

7.3.1 CONFIGURAÇÕES

A máquina virtual possui as seguintes configurações:

Quadro 8: Software e hardware da máquina virtual.

Sistema Operacional	Debian 8 Jessie x86_64
Processador	Intel(R) Core(TM) i5-4670 CPU @ 3.40GHz
Memória RAM	8Gb DDR3 @ 1600MHz
Disco rígido	100Gb SATA Control

Fonte: O Autor.

7.4 BANCO DE DADOS

Será utilizado a mesma máquina virtual para execução dos dois bancos de dados.

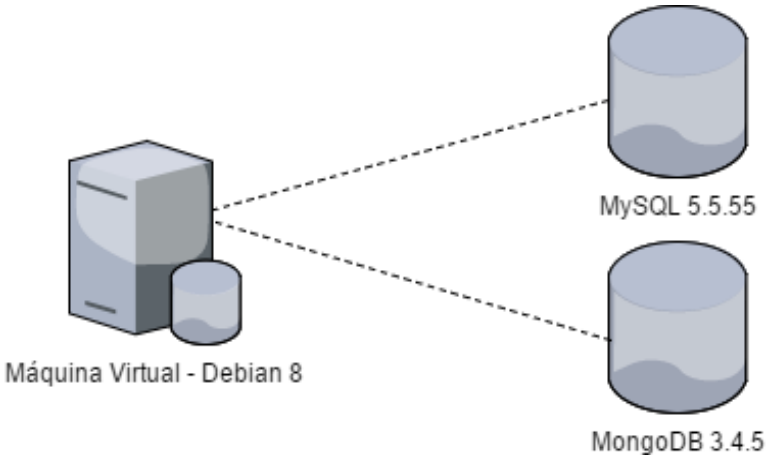
Quadro 9: Versão dos SGBD utilizados.

SGBD	Versão
MySQL	v5.5.55
MongoDB	v3.4.5

Fonte: O Autor.

Na figura 10 segue um diagrama que representa a estrutura da máquina virtual, contendo ambos os bancos de dados:

Figura 10: Diagrama da máquina virtual executando simultaneamente duas bases de dados.



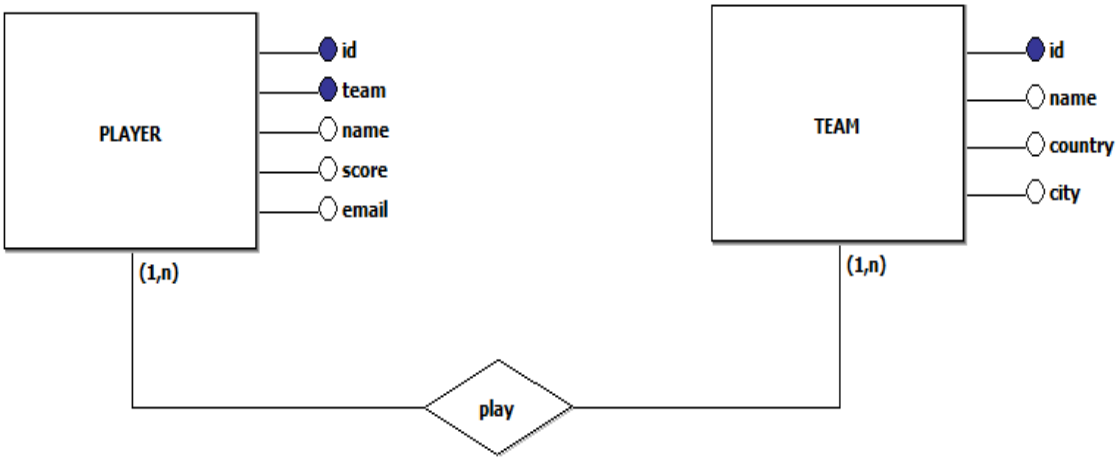
Fonte: O Autor.

7.5 MYSQL

7.5.1 DIAGRAMA DE ENTIDADE E RELACIONAMENTOS

Neste trabalho será utilizado o seguinte diagrama de entidade e relacionamento para realização dos testes:

Figura 11: Diagrama de entidade e relacionamento do banco de dados relacional.



Fonte: O Autor.

O diagrama consiste em duas entidades, jogadores e times. Um jogador pode estar atrelado a um ou vários times e um time pode estar vinculado a um ou vários jogadores. Este vinculo é estabelecido pela chave estrangeira especificada na tabela “player”, denonima “team”, que contém o código identificador(chave primária) do time.

7.6 MONGODB

7.6.1 DOCUMENTO JSON

Na figura 12 segue um exemplo de estrutura dos documentos, utilizados no MongoDB, para este trabalho:

Figura 12: Documento JSON.


```
{
  "_id": ObjectId("594c0c2eaf332cb785b4a269"),
  "name": "Guilherme de Pontes",
  "email": "teste@teste.com.br",
  "score": 742,
  "team": [
    {
      "name:": "team1",
      "country": "Brasil",
      "city": "Sao Paulo"
    },
    {
      "name:": "team2",
      "country": "Argentina",
      "city": "Buenos Aires"
    },
    {
      "name:": "team3",
      "country": "Estados Unidos",
      "city": "Orlando"
    }
  ]
}
```


Fonte: O Autor

Neste documento exemplo, o jogador e os times estão todos armazenados em apenas uma única *collection*, pois diferente do modelo relacional, o MongoDB não trabalha com relacionamentos.

7.7 PROCEDIMENTO DO TESTE

Para comparar o desempenho entre os bancos de dados, será executado *querys* de inserção, consulta e exclusão em massa utilizando um *script* que apresenta o tempo preciso de execução das *querys*, com o intuito de identificar quanto tempo foi necessário aguardar para a conclusão do processo. Com estes resultados será possível elaborar gráficos que ilustram o desempenho obtido entre cada SGBD.

7.8 SÉRIE DE TESTE

O *script* será programado para executar *querys* de 100, 1.000, 100.000 e 1.000.000 de registros simultaneamente. Desta forma será coletado resultados aleatórios, pois com esta variação na quantidade dos registros, o tempo para execução das *querys* poderá sofrer mudança de comportamento.

7.9 PROGRAMA PARA TESTE

Para realização dos testes, foi desenvolvido um *script* utilizando a plataforma de desenvolvimento NodeJS. Com este script é possível obter resultados precisos em milisegundos, por cada *query* executada.

7.10 MÉTODO DE REALIZAÇÃO DO TESTE

Os scripts foram desenvolvidos da seguinte forma:

Figura 13: Código fonte do index.js

```
var spinner = require("char-spinner"); //SPINNER DE LOADING
var async = require("async"); //MODULO async PARA TRABALHAR COM JAVASCRIPT ASSINCRONO
spinner();

var dataSize = 1000000; //DEFINE QUANTIDADE DE QUERYS A SEREM EXECUTADAS

async.series([
    //FUNCAO DE INSERCAO QUE EXECUTA O ARQUIVO mongo.js E mysql.js
    function(callback){
        require('./mongo')('insert',dataSize,callback);
    },
    function(callback){
        require('./mysql')('insert',dataSize,callback);
    },

    //FUNCAO DE BUSCA QUE EXECUTA O ARQUIVO mongo.js E mysql.js
    function(callback){
        require('./mongo')('find',dataSize,callback);
    },
    function(callback){
        require('./mysql')('find',dataSize,callback);
    },

    //FUNCAO DE EXCLUSAO QUE EXECUTA O ARQUIVO mongo.js E mysql.js
    function(callback){
        require('./mongo')('remove',dataSize,callback);
    },
    function(callback){
        require('./mysql')('remove',dataSize,callback);
    }

],function(){ //FIM DA EXECUCAO, FINALIZA O PROCESSO
    process.exit();
})
```

Fonte: O Autor.

Neste primeiro *script*, denominado “index.js”, está sendo definido a quantidade de *querys* a serem executadas utilizando a variável “dataSize” e definindo também as funções que executam os *scripts* “mongo.js” e “mysql.js”.

Houve a necessidade de separar os *scripts*, para que não ocorra conflito de conexão entre os bancos de dados. Desta forma também é possível tornar independente o tempo de execução de cada SGBD, pois um está obrigatoriamente separado do outro.

O módulo *spinner* é apenas visual, pois quando for executado o *script*, ficará uma barra que ilustra um processo de *loading*.

Já o módulo *async* é utilizado para programação assíncrona. A programação assíncrona significa que o código é executado em um ciclo de eventos. Quando uma operação de bloqueio é necessária, a solicitação é iniciada e o código continua funcionando sem bloquear o resultado. Quando a resposta está pronta, uma interrupção é disparada, o que faz com que um manipulador de eventos seja executado, onde o fluxo de controle continua. Desta forma, um único segmento de programa pode lidar com muitas operações simultâneas.

O *script* que dispara *querys* no MongoDB encontra-se em apêndice(Apêndice A), pois é um código extenso.

Para realizar a inserção dos dados, foi necessário utilizar uma técnica denominada *promise*.

A ideia central por trás das *promises* é que a mesma representa o resultado de uma operação assíncrona. Uma *promise* enquadra-se em um dos três estados diferentes:

- Pendente - Estado inicial de uma *promise*.
- Concluído - Estado de uma *promise* que representa uma operação bem sucedida.
- Rejeitado - O estado de uma *promise* que representa uma operação com falhas.

Uma vez que uma promessa é cumprida ou rejeitada, é imutável, ou seja, nunca pode mudar novamente.

Para popular o banco de dados com informações aleatórias, foi utilizado o método Faker.

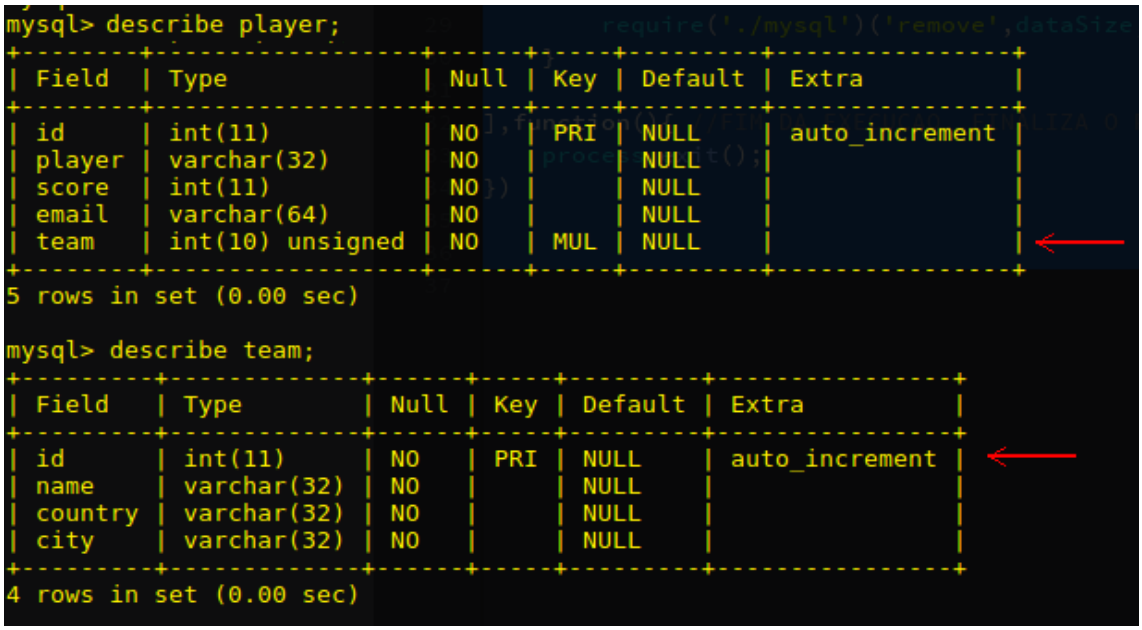
Este *script* consiste de basicamente 3 etapas:

- *Promise* objetosParaAdd, que é usado apenas no processo de inserção dos dados.
- Função de inserção – self.insert.
- Função de busca – self.find.
- Função de exclusão – self.remove.

O *script* que executa funções no banco de dados MySQL, encontra-se em apêndice(Apêndice B). Este *script* segue o mesmo conceito e estrutura do “mongo.js”, porém, o código acabou ficando maior pois o MySQL utiliza uma lógica diferente do MongoDB e necessita que seja criado tabelas, definindo o atributo de cada campo.

Outro detalhe está no relacionamento das tabelas *player* e *team*. O MySQL trabalha com relacionamentos, diferentemente do MongoDB, onde todas as informações encontram-se em apenas uma coleção.

Figura 14: Relacionamento das tabelas *Player* e *Team*(MySQL).



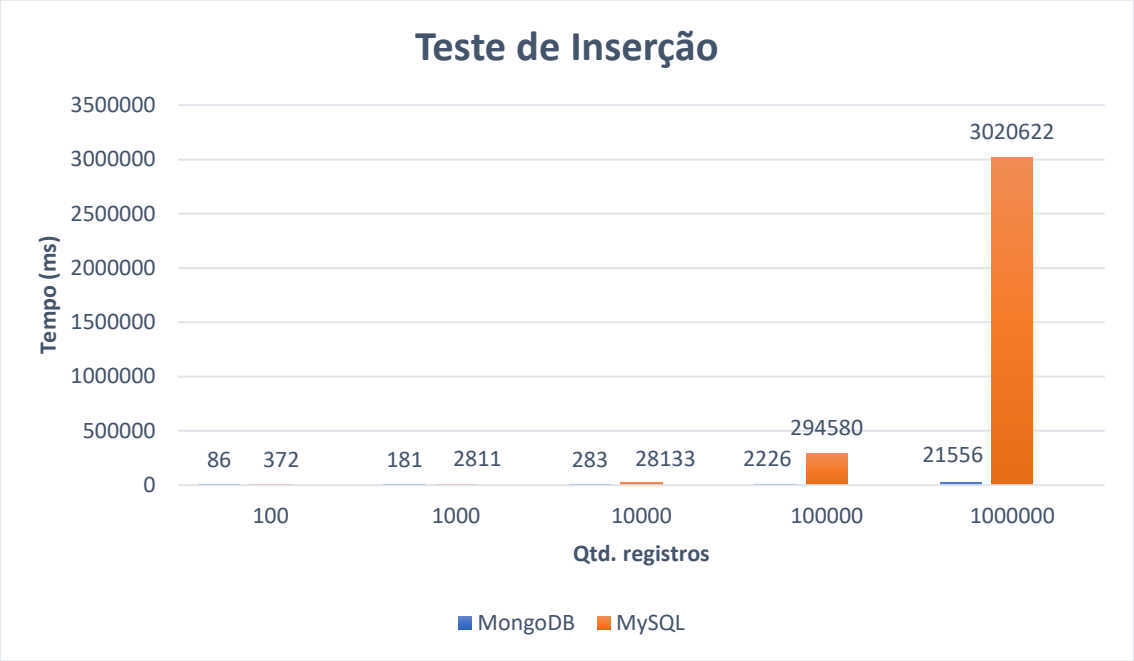
Fonte: O Autor

7.11 COMPORTAMENTO DO TESTE

Neste capítulo, serão apresentados os resultados dos testes de desempenho, em forma de tabelas e gráficos.

De antemão, segue abaixo o gráfico que ilustra o teste de inserção, com a média de tempo para concluir a execução de cada *query*. Foram realizados testes de 100, 1.000, 10.000, 100.000 e 1.000.000 de inserções simultaneamente:

Figura 15: Gráfico em função do teste de inserção.

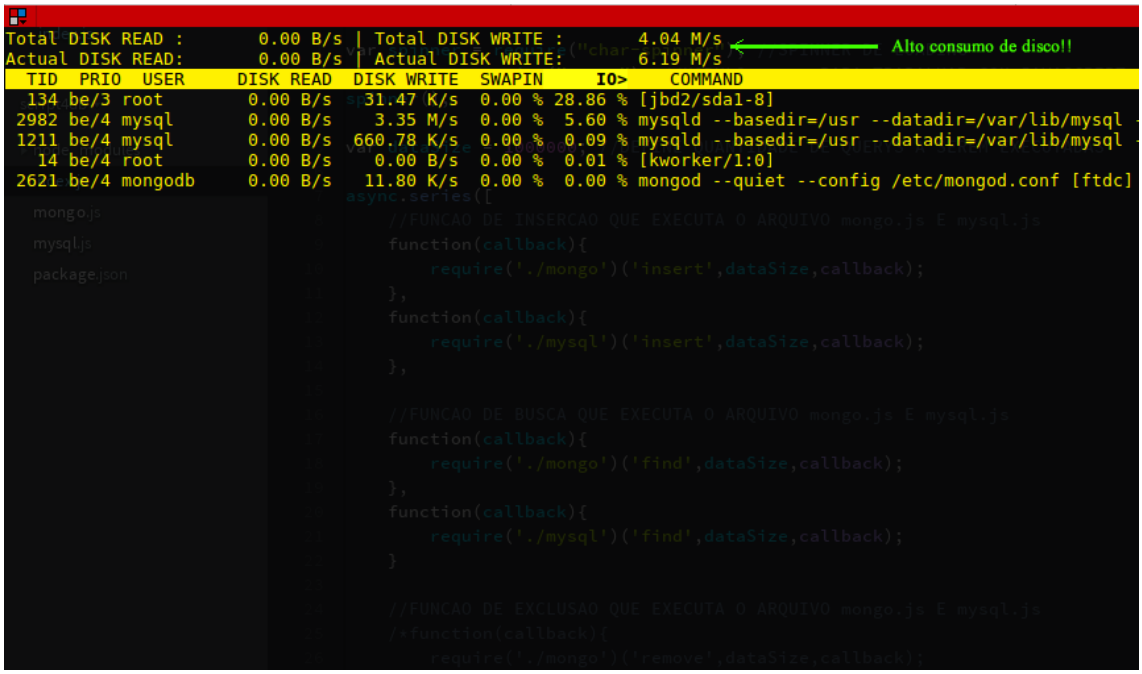


Fonte: O Autor.

O alto desempenho do MongoDB, perante os testes de inserção é notório e os resultados se mantém abaixo da média do MySQL.

Além do alto custo de tempo para inserir as informações, o MySQL utiliza 100% do disco para cumprir a tarefa de 1.000.000 de registros, gerando lentidão em aplicações que estão em execução, como por exemplo o Navegador de Internet, abrir e navegar em pastas do sistema, etc.

Figura 16: Consumo de disco excessivo ao inserir informações em massa no MySQL.



Fonte: O Autor.

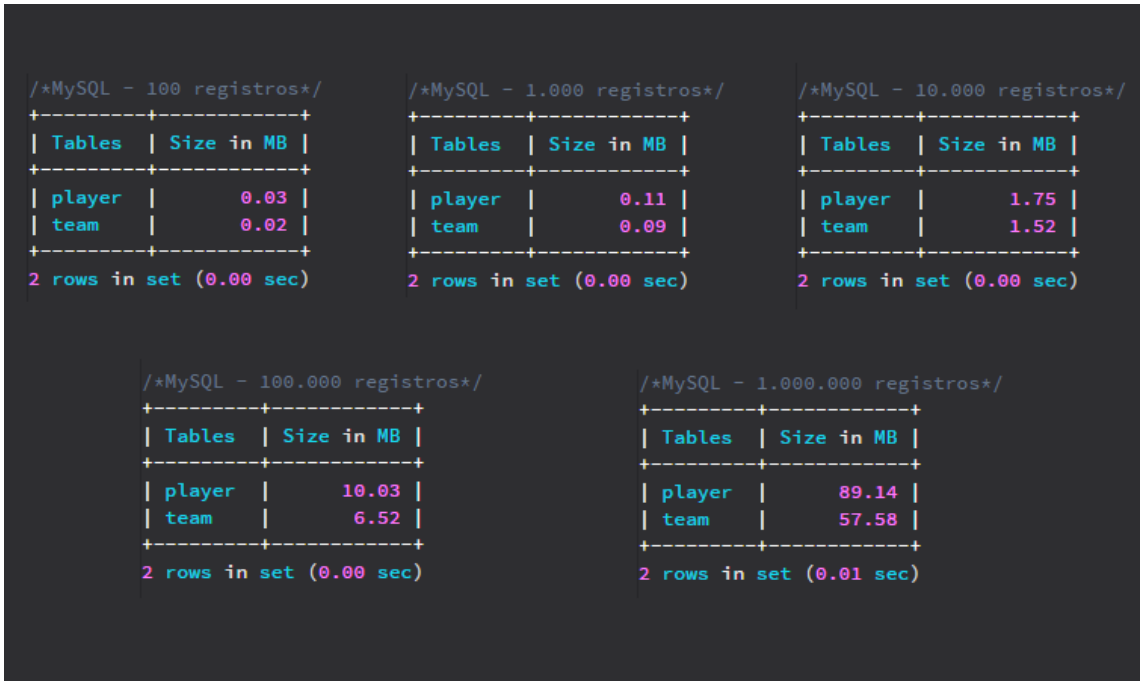
Além do alto consumo de processamento do disco, o MySQL acaba perdendo também no consumo de memória física:

Figura 17: Armazenamento de memória física – MongoDB.

```
/*MongoDB - 100 registros*/
vboxdeb(mongod-3.4.5) test> show collections
player → 0.017MB / 0.004MB

/*MongoDB - 1.000 registros*/
vboxdeb(mongod-3.4.5) test> show collections
player → 0.168MB / 0.109MB

/*MongoDB - 10.000 registros*/
vboxdeb(mongod-3.4.5) test> show collections
player → 1.682MB / 1.012MB

/*MongoDB - 100.000 registros*/
vboxdeb(mongod-3.4.5) test> show collections
player → 16.810MB / 10.063MB

/*MongoDB - 1.000.000 registros*/
vboxdeb(mongod-3.4.5) test> show collections
player → 168.123MB / 83.383MB
```

Fonte: O Autor.

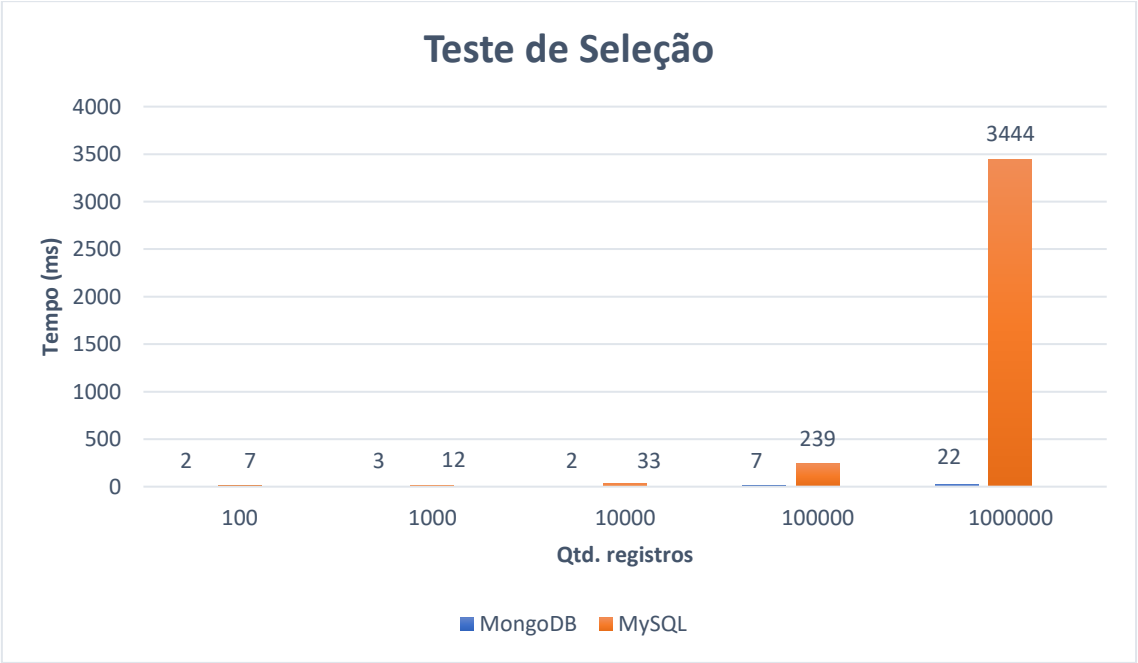
Figura 18: Armazenamento de memória física – MySQL.



Fonte: O Autor.

Com relação ao teste de busca, o MongoDB também se sobressai no tempo de resposta e apresenta as informações contidas na collection quase que instantaneamente, enquanto o MySQL, demonstra um certo delay:

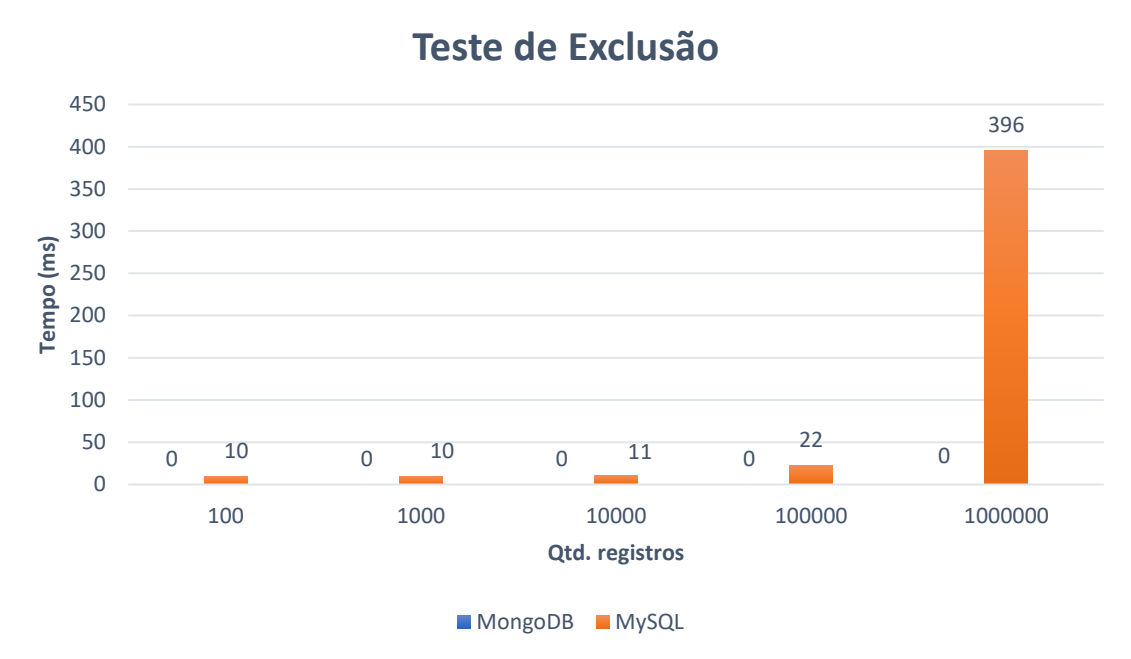
Figura 19: Gráfico em função do teste de busca.



Fonte: O Autor.

Por fim, o teste de exclusão também concede os pontos ao MongoDB, pois ao executar a *query* de exclusão, independente da quantidade de registros, é apresentado a resposta instantaneamente, em surpreendentes 0ms:

Figura 20: Gráfico em função do teste de exclusão.



Fonte: O Autor.

8. CONCLUSÃO E TRABALHOS FUTUROS

No trabalho desenvolvido, foi levado em consideração conceitos gerais sobre o paradigma de armazenamento de informações NoSQL. Sendo assim, houve a necessidade de realizar um estudo aprofundado sobre o SGBD MongoDB, que analisou sua estrutura de dados e métodos de armazenamento.

Com relação às informações que foram adquiridas durante os estudos e pesquisas, foi desenvolvido um script utilizando a plataforma de desenvolvimento NodeJS que insere, busca e remove informações em ambos os bancos de dados, relacional e não-relacional, com a finalidade de identificar até que ponto o MongoDB se sobressai perante o MySQL.

Este projeto tem como objetivo geral apresentar resultados reais, de quanto tempo cada *query* demora para ser concluída e motivar o desenvolvedor a levar em consideração tecnologias que estão a total disposição para qualquer programador usufruir e evitar futuros problemas de falta de desempenho na aplicação, afetando o usuário final.

Por fim, os resultados coletados nos testes foram extremamente satisfatórios, pelo fato de provar que realmente o MongoDB é um banco de alta performance em relação ao MySQL. Pôde-se observar que conforme a massa de dados aumentava, o MongoDB adquire mais vantagem em relação ao MySQL, tanto na busca, inserção e exclusão dos dados. O alto desempenho é notório à partir de 100.000(cem mil) registros, pois é neste momento que o MySQL exige um alto custo de tempo para concluir suas operações.

Propostas futuras com relação a este tema, tem como objetivo criar um ambiente híbrido, utilizando o MySQL em paralelo com o MongoDB, pois desta forma será eliminado as desvantagens de cada SGBD e, subentende-se que a aplicação ganhará desempenho e consistência ao manipular as informações armazenadas na base dados.

9. REFERÊNCIAS

Baidu and MongoDB. Disponível em <https://www.mongodb.com/blog/post/mongodb-at-baidu-powering-100-apps-across-600-nodes-at-pb-scale> Acesso em: 20 Agosto 2017.

CARPENTER, Jeff; HEWITT, Eben. Cassandra: The Definitive Guide; 2 ed. California:O'Reilly, 2016.

CHAMBERLIN, Donald. A Complete Guide to DB2: Universal database; San Francisco:Morgan Kaufmann, 1998.

CODD, Edgar F. The Relational Model For Database Management; 2 ed. Boston:Addison-Wesley, 1990.

DATE, Christopher J. The Database Relational Model: A Retrospective Review and Analysis; Boston:Addison Wesley, 2001.

Difference between scaling horizontally and vertically for databases. Disponível em <https://stackoverflow.com/questions/11707879/difference-between-scaling-horizontally-and-vertically-for-databases> Acesso em: 7 Abril 2017.

FOWLER, Adam. NoSQL For Dummies; New Jersey:Wiley, 2015.

HOWS, David et al. The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB; 3 ed. New York:Apress, 2015.

Introdução a Banco de Dados. Disponível em <https://www.ime.usp.br/~jef/apostila.pdf> Acesso em: 14 abril 2017.

MongoDB Documentation. Disponível em <https://docs.mongodb.com/getting-started/shell> Acesso em: 7 Abril 2017.

MySQL Documentation. Disponível em <https://dev.mysql.com/doc> Acesso em: 7 Abril 2017.

Relational Database. Disponível em <http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/reldb/> Acesso em: 7 abril 2017.

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

SYED, Basarat. Beginning Node.JS; New York:Apress, 2014.

TAURION, Cezar. Big Data; Rio de Janeiro:Brasport, 2013.

TAYLOR, Allen G. SQL para Leigos: For Dummies; Rio de Janeiro:Alta Books, 2016.

VAISH, Gaura. Getting Started with NoSQL; Birmingham:Packt, 2013.

WATSON, Paul. MySQL - A Beginner's Guide; California:CreateSpace, 2016.

WIESE, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases; Berlim:Walter de Gruyter, 2015.

10. APÊNDICES

APÊNDICE A – MONGO.JS

```
var async = require('async'); //Módulo assincrono, pois o código precisa ser
assincrono
var faker = require('Faker'); //Módulo Faker para gerar informações aleatórias na
função de inserção
var mongojs = require('mongojs'); //Biblioteca para mapear informações no banco de
dados
var db = mongojs('test',["player"]); //Definindo a tabela e a coleção que será
utilizada

var self = {}; //Variável self para devolver as informações para a função mestre >
index.js

var MongoClient = require('mongodb').MongoClient; //Instância a conexão com o banco de
dados
var url = "mongodb://localhost:27017/test"; //Instância o caminho do banco de dados e
porta default

//Instância os objetos para adicionar no banco de dados, utilizando promises
var objetosParaAdd = function(dataSize) {
    //Se tudo ocorre bem, retorne RESOLVE caso contrário, REJECT!
    return new Promise(function (resolve, reject){
        var i = 0; //Inicia i em 0
        var teste = {} //Variável teste que receberá a estrutura da coleção com
informações fakes
        var arr = [] //Variável arr que válida se está tudo ocorrendo bem
        while(i < dataSize){ //Enquanto i for menor que dataSize, continue com o loop
             teste = {name:faker.Name.findName(), email:faker.Internet.email(),
                      score:Math.floor(Math.random()*1000),
                        team: [{name:faker.Company.companyName().split("
")[0].split(",")[0],
                        country:faker.Address.ukCountry(),
                        city:faker.Address.city()}]};

            arr.push(teste)
            i++; //i sendo incrementado

        }
        resolve(arr) //Se arr estiver sem erros, código resolvido
        reject(function(err){ //Caso contrário o código será rejeitado
            console.log('CODE REJECT!', err)
        })
    })
}

self.insert = function(dataSize,done){ //Função para inserir informações no banco de
dados
    var run = []; //Variável run para executar informações no banco de dados

    MongoClient.connect(url, function(err, db) { //Estabelecendo conexão com o banco
        if (err) throw err;

        objetosParaAdd(dataSize).then(function(data){ //objetosParaAdd passando como
parâmetro dataSize
            console.time('mongo insert'); //Inicia a contagem do tempo

            db.collection("player").insert( data ) //Insere a informações dentro de
data

            if(done) done();
            console.timeEnd('mongo insert'); //Finaliza a contagem do tempo

        }).catch(function (err){
            console.log(err)
        })
    });
}

self.find = function(dataSize,done){ //Função para buscar informações no banco de
dados
```

```
    var run = [];

    MongoClient.connect(url, function(err, db) { //Estabelecendo conexão com o banco
        db.collection('player', function(err, collection) { //Função que define qual
tabela será executado a busca
            collection.find(); //Executa a busca
        });
    });

    console.time('mongo select'); //Inicia a contagem do tempo
    async.series(run,function(err,data){
        console.timeEnd('mongo select'); //Finaliza a contagem do tempo
        if(done) done();
    });
}

self.remove = function(dataSize,done){ //Função para remover informações no banco de
dados
    var run = [];

    MongoClient.connect(url, function(err, db) { //Estabelecendo conexão com o banco
        db.collection('player', function(err, collection) { //Função que define qual
tabela será executado a remoção
            collection.remove({}); //Executa a remoção
        });
    });

    console.time('mongo remove'); //Inicia a contagem do tempo
    async.series(run,function(err,data){
        console.timeEnd('mongo remove'); //Finaliza a contagem do tempo
        if(done) done();
    });
}

module.exports = function(type,dataSize,done){ //Retornar informações para a função
mestre > index.js
    return self[type](dataSize,done);
}
```

APÊNDICE B – MYSQL.JS

```
var async = require('async'); //Módulo assicrono, pois o código precisa ser
assincrono
var faker = require('Faker'); //Módulo Faker para gerar informações
aleatórias na função de inserção

var self = {}; //SELF SERA UTILIZADO PARA MANTER A REFERENCIA ORIGINAL, OU
SEJA, index.js

self.insert = function(dataSize,done){ //Função para inserir informações no
banco de dados
    var mysql = require('mq-node')({
        host     : 'localhost',
        user     : 'root',
        password : '123',
    });

    var run = [];

    run.push(function(callback){
        mysql.connection.connect(function(err){
            callback();
        });
    })

    run.push(function(callback){
        mysql.query('CREATE DATABASE IF NOT EXISTS test',function(data,err){
            callback();
        });
    })

    run.push(function(callback){
        mysql.connection.changeUser({database:'test'},function(data,err){
            callback();
        });
    })

    run.push(function(callback){
        mysql.query('DROP TABLE test',function(data,err){
            callback();
        });
    });

    run.push(function(callback){
        mysql.query('DROP TABLE team',function(data,err){
            callback();
        });
    });

    run.push(function(callback){
        mysql.query('CREATE TABLE IF NOT EXISTS test ('+
            'id int(11) NOT NULL AUTO_INCREMENT,'+
            'player varchar(32) NOT NULL,'+
            'score int(11) NOT NULL,'+
            'email varchar(64) NOT NULL,'+
            'team int(10) unsigned NOT NULL,'+
            'PRIMARY KEY (id),'+
            'KEY team (team)'+
        ')',function(data,err){
            callback();
        });
    });

    run.push(function(callback){
        mysql.query('CREATE TABLE IF NOT EXISTS team ('+
            'id int(11) NOT NULL AUTO_INCREMENT,'+
            'name varchar(32) NOT NULL,'+
```

```
            'country varchar(32) NOT NULL,'+
            'city varchar(32) NOT NULL,'+
            'PRIMARY KEY (id)'+
        ')',function(data,err){
            callback();
        });
    });

    for(var i=0;i<dataSize;i++){
        run.push(function(callback){
            mysql.insert('test',{
                player:faker.Name.findName(),
                email:faker.Internet.email(),
                team:Math.floor(Math.random()*dataSize),
                score:Math.floor(Math.random()*1000)
            },function(data,err){
                callback();
            });
        });

        run.push(function(callback){
            mysql.insert('team',{
                city:faker.Address.city(),
                country:faker.Address.ukCountry(),
                name:faker.Company.companyName().split(" ")[0].split(",")[0]
            },function(data,err){
                callback();
            });
        });
    }

    console.time('mysql insert')
    async.series(run,function(err,data){
        console.timeEnd('mysql insert');
        mysql.end();
        if(done) done();
    });
}

self.find = function(dataSize,done){ //Função para buscar informações no
banco de dados
    var mysql = require('mq-node')({
        host     : 'localhost',
        user     : 'root',
        password : '123',
    });

    var run = [];

    run.push(function(callback){
        mysql.connection.changeUser({database:'test'},function(data,err){
            callback();
        });
    });

    run.push(function(callback){
        mysql.select({
            from:['test as t1','team as t2'],
            cols:['player','name'],
            where:'t2.id=t1.team'
        },function(data,err){
            for(var i in data) if(!data[i]['name']) console.log('mysql error
: team no name');
            callback();
        })
    });

    console.time('mysql select')
```

```
    async.series(run,function(err,data){
        console.timeEnd('mysql select');
        mysql.end();
        if(done) done();
    });
}

self.remove = function(dataSize,done){ //Função para remover informações no
banco de dados
    var mysql = require('mq-node')({
        host     : 'localhost',
        user     : 'root',
        password : '123',
    });

    var run = [];

    run.push(function(callback){
        mysql.connection.changeUser({database:'test'},function(data,err){
            callback();
        });
    });

    run.push(function(callback){
        mysql.query('TRUNCATE TABLE test',function(data,err){
            callback();
        });
    });

    run.push(function(callback){
        mysql.query('TRUNCATE TABLE team',function(data,err){
            callback();
        });
    });

    console.time('mysql remove')
    async.series(run,function(err,data){
        console.timeEnd('mysql remove');
        mysql.end();
        if(done) done();
    });
}

module.exports = function(type,dataSize,done){ //Retornar informações para a
função mestre > index.js
    return self[type](dataSize,done);
}
```