INSTITUTO FEDERAL DE SÃO PAULO - IFSP

ÁREA DE INFORMÁTICA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - ADS

WILLIAM DE JESUS OLIVEIRA

CAIO RODRIGO CARDOSO SOARES

MONGODB X SQL SERVER 2012: UMA COMPARAÇÃO ENTRE SISTEMAS GERENCIADORES DE BANCOS DE DADOS RELACIONAIS E A TECNOLOGIA NOSQL

TRABALHO DE CONCLUSÃO DE CURSO - TCC

CARAGUATATUBA

2014

WILLIAM DE JESUS OLIVEIRA
CAIO RODRIGO CARDOSO SOARES

MONGODB X SQL SERVER 2012: UMA COMPARAÇÃO ENTRE SISTEMAS GERENCIADORES DE BANCOS DE DADOS RELACIONAIS E A TECNOLOGIA NOSQL

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo, da Área de Informática, do Instituto Federal de São Paulo.

Orientador:
Prof. Dr. Lineu Fernando Stege Mialaret

CARAGUATATUBA

2014

O676m Oliveira, William de Jesus

MongoDB X SQL Server 2012: uma comparação entre sistemas gerenciadores de bancos de dados relacionais e a tecnologia NoSQL / William de Jesus Oliveira; Caio Rodrigo Cardoso Soares.

101 f.

Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de São Paulo – IFSP, Caraguatatuba, SP, 2014.

1. Administração de Bancos de Dados. 2. Comparativo de Ferramentas. 3. Workbench. I. MongoDB X SQL Server 2012: uma comparação entre sistemas gerenciadores de bancos de dados relacionais e a tecnologia NoSQL

CDD: 005.74

Ministério da Educação
Instituto Federal de São Paulo
Campus Caraguatatuba

Nome do Diretor
Nome do Coordenador
Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

MONGODB X SQL SERVER 2012: UMA COMPARAÇÃO ENTRE SISTEMAS GERENCIADORES DE BANCOS DE DADOS RELACIONAIS E A TECNOLOGIA NOSQL

por

WILLIAM DE JESUS OLIVEIRA
CAIO RODRIGO CARDOSO SOARES

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 15 de setembro de 2014 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas (ADS). Os candidatos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados, a qual após deliberação considerou o trabalho aprovado.

_____
Lineu Fernando Stege Mialaret
Professor Orientador

_____
Prof. Henrique Gonçalves Salvador
Presidente

_____
Prof. Lucas Venezian Povoa
Membro

Dedicamos este trabalho às nossas famílias pelo apoio que nos foi dado durante o desenvolvimento do mesmo.

AGRADECIMENTOS

Certamente estes parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase de nossas vidas. Portanto, desde já pedimos desculpas àquelas que não estão presentes entre essas palavras, mas elas podem estar certas que fazem parte do nosso pensamento e de nossa gratidão.

Agradecemos ao nosso orientador, Prof. Dr. Lineu Fernando Stege Mialaret, pela sabedoria com que nos guiou nesta trajetória e por toda a sua paciência para conosco.

Ao Prof. Me. Nelson Alves Pinto, pela sua prontidão em oferecer os recursos e equipamentos necessários para a realização deste trabalho.

Aos nossos colegas de sala que durante todo o período no qual estivemos juntos, estavam sempre nos ajudando a superar as dificuldades enfrentadas durante o trajeto até aqui.

Gostaríamos de deixar registrado também, o nosso reconhecimento à nossa família, pois acreditamos que sem o apoio deles seria muito difícil vencer esse desafio.

Enfim, a todos os que por algum motivo contribuíram para a realização desta pesquisa.

O crescimento dos bancos de dados NoSQL marca o fim da era de domínio dos bancos de dados relacionais. Mas, os bancos de dados NoSQL não serão os novos dominantes. Os relacionais ainda serão populares e usados na maioria das situações. Contudo, não serão mais a escolha automática. A era da persistência poliglota começou.

(FOWLER, Martin, 2013)

RESUMO

A cada dia cresce o volume de informações empresariais, e com isso aumenta também o número de organizações que optam por utilizar soluções NoSQL (*Not only SQL*) ao invés de Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR). Esta troca é normalmente justificada pela dificuldade de se adaptar a realidade em tabelas com estruturas rígidas e, pela dificuldade na manipulação de grandes quantidades de dados em um SGBDR, enquanto que, os questionamentos acerca dos sistemas NoSQL são na maioria das vezes pela sua inconsistência e pela impossibilidade de fazer junções entre registros em unidades de armazenamento separadas. O objetivo desta pesquisa é a comparar um SGBDR e um SGBD NoSQL, no caso SQL Server e o MongoDB respectivamente, em um cenário determinado. Para a realização dos testes foi utilizada uma ferramenta de teste específica, possibilitando a identificação das situações que melhor se adaptam a cada um destes sistemas gerenciadores.

Palavras-Chave: MongoDB. SQL Server. JMeter. NoSQL. Comparação.

ABSTRACT

The volume of business information is growing every day and, it increases the number oforganizations that choose to use NoSQL (Not only SQL) solutions instead of Relational Database Management Systems (RDBMS). This exchange is usually justified by the difficulty of adapting the reality into tables with rigid structures and the difficulty in handling large amounts of data in an RDBMS, whereas, the questions about NoSQL systems are mostly by their inconsistency and the inability to perform joins (Inner Join) between records in separate storages. The objective of this research is to compare a Relational and a NoSQL DBMS, SQL Server and MongoDB respectively, in a specific scenario. For the tests, a specific tool for testing was used, enabling the identification of situations that are best suited to each of these management systems.

Keywords: *MongoDB. SQL Server. JMeter. NoSQL. Comparison.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

LISTA DE ABREVIATURAS

NoSQL	*Not Only* SQL
T-SQL	*Transact*-SQL
BI	*Business Inteligence*
RAM	*Random Access Memory*
HD	*Hard Disk*
DVD	*Digital Video Disk*

LISTA DE SIGLAS

SGBD	Sistema Gerenciador de Bancos de Dados
SGBDR	Sistema Gerenciador de Bancos de Dados Relacionais
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
SQL	*Structured Query Language*
ANSI	*American National Standards Institute*
DML	*Data Manipulation Language*
DDL	*Data Definition Language*
DCL	*Data Control Language*
DQL	*Data Query Language*
DTL	*Data Transaction Language*

LISTA DE ACRÔNIMOS

MS	Microsoft Corporation®
JSON	*JavaScript Object Notation*
BSON	*Binary* JSON
XML	*Extended Mark-up Language*
BASE	*Basically Available, Soft state, Eventual consistency*
TAG	*Metadata*

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Atualmente, com a grande quantidade de dados produzidos diariamente pelas organizações, a manipulação e tratamento dos mesmos vêm sendo cada vez mais complexa. Algumas empresas, dentre elas, a Google, o Facebook e o Twitter têm optado pela utilização de Bancos de Dados não relacionais para armazenar suas informações, devido à relativa facilidade de manipulação destas em comparação com o uso de Sistemas Gerenciadores de Bancos de Dados Relacionais (SGBRs).

A justificativa de tal atitude se dá na maioria das vezes pelo fato de que bancos de dados não relacionais não possuem uma estrutura fixa de armazenamento, tal como ocorre nos bancos de dados relacionais.

Nos bancos de dados relacionais cada tabela possui um número fixo de colunas e um tipo de dado que pode ser armazenado. Por este motivo, quando não existe um dado para ser inserido em uma coluna da tabela, esta fica com um valor nulo ocupando espaço em vão na mídia de armazenamento.

Já na tecnologia não relacional, também conhecida como *Not Only SQL* (NoSQL), a estrutura de armazenamento não tem um número mínimo e nem máximo de elementos que podem ser inseridos na “coleção” (designação comum que se refere ao conjunto de objetos, documentos e outras unidades de armazenamento de um banco de dados NoSQL). Ou seja, o espaço consumido para o armazenamento de uma unidade de informação é correspondente ao número e tamanho dos dados inseridos nesta unidade.

Outro motivo da preferência destas organizações pelo uso de tecnologias NoSQL se dá ao fato da velocidade de recuperação das informações armazenadas nas bases de dados. Em um banco de dados relacional é comum à distribuição das informações de um único registro em diversas tabelas visando evitar ao máximo o armazenamento de valores nulos, porém, esta técnica tem o seu ponto fraco na

recuperação das informações, onde é gasto muito processamento para a obtenção dos dados de um registro, uma vez que se encontram espalhados.

Por outro lado, nos bancos de dados NoSQL devido a flexibilidade de uma unidade de armazenamento consumir apenas o tamanho de suas informações e estas não possuírem valores nulos, torna-se possível realizar a inserção em apenas uma unidade e obter estas mesmas informações vindas de um único lugar. Em outras palavras, torna-se muito mais rápido encontrar o que se está procurando em meio a um grande volume de informações.

1.2 OBJETIVOS DA PESQUISA

O objetivo geral deste trabalho é a comparação de desempenho entre SGBDRs e a tecnologia NoSQL. São utilizados o SGBD Relacional SQL Server em sua versão 2012 e o MongoDB (Banco de dados NoSQL orientado a documentos). A comparação é feita com base nos aspectos: teste de carga e *stress* e uso de recursos de hardware. Os testes de carga e *stress* serão feitos com a ferramenta JMeter.

O teste de carga visa verificar a latência e a vazão (*throughput*), já o teste de *stress,* apesar de muitas vezes ser feito concomitantemente, visa verificar como o gerenciador se comporta em condições anormais de uso.

A verificação do consumo de hardware é feita por meio do Gerenciador de Tarefas do Windows na aba “Desempenho”, onde são anotados o consumo mínimo e o máximo durante os testes e calculada a média aritmética.

1.3 RESULTADOS ESPERADOS

Ao final deste trabalho são esperados resultados com informações capazes de auxiliar técnicos, administradores de bancos de dados e analistas e/ou desenvolvedores na tomada de decisões quanto ao uso da tecnologia que oferece

melhores recursos e desempenho para cada cenário confrontado no cotidiano de trabalho.

1.4 ESTRUTURA DO TRABALHO

No Capítulo 2 serão apresentados os conceitos de Banco de Dados, abordando tanto o modelo relacional quanto o modelo não relacional, além de, também serem apresentados os dois gerenciadores utilizados neste trabalho, o MS SQL Server e o MongoDB.

No Capítulo 3 é apresentada uma comparação entre as sintaxes utilizadas em cada um dos gerenciadores nas operações básicas de inserção, seleção, atualização e remoção.

No Capítulo 4 são apresentadas as instalações dos dois gerenciadores, mostra-se também a ferramenta de testes Apache Jmeter e expõem-se todos os testes de carga e *stress* realizados.

Finalmente, no Capítulo 5 apresenta-se a conclusão final, bem como as perspectivas para trabalhos futuros.

2 BANCO DE DADOS

2.1 INTRODUÇÃO A BANCO DE DADOS

> *"Um sistema de gerenciamento de banco de dados (DBMS) é uma coleção de dados inter-relacionados e um conjunto de programas para acessar estes dados. A coleção de dados normalmente chamada de banco de dados, contem informações relevantes a uma empresa. O principal objetivo de um DBMS é fornecer uma maneira de recuperar informações de banco de dados que seja tanto conveniente quanto eficiente. "(SILBERSCHATZ; KORTH; SUDARSHAN, 2006, p. 1).*

Antes de existirem os Sistemas Gerenciadores de Bancos de Dados (SGBDs), as informações eram armazenadas em arquivos do sistema operacional. Para cada operação era feita uma aplicação específica que era responsável por manipular e salvar estes dados. O principal problema deste método é que se mudasse o sistema operacional ou a aplicação, provavelmente estes arquivos seriam perdidos ou no mínimo prejudicados. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006, p. 2-3)

A cada nova necessidade de manipulação de dados, se fazia necessária a criação de uma nova aplicação que fizesse este controle. Além disso, toda restrição era tratada apenas na aplicação o que aumentava os riscos de problemas como redundância e inconsistência, integridade, dificuldade de acesso e principalmente, a segurança da informação. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006, p. 3)

A cada dia cresce a quantidade de dados que cada empresa armazena, e o acesso rápido a estas informações são fatores críticos do sucesso, por isso foi preciso criar sistemas que gerenciassem estas informações tornando assim este procedimento mais fácil. Com base neste conceito foram criados os SGBDs, para auxiliar no armazenamento e tratamento de toda informação que precisasse ser persistida.

2.2 SISTEMA DE GERENCIAMENTO DE BANCOS DE DADOS RELACIONAIS

Basicamente, Sistemas Gerenciadores de Bancos de Dados Relacionais (SGBDRs) são coleções de dados e um conjunto de sistemas computacionais construídos tendo por base o modelo relacional e que têm a finalidade de gerenciar bancos de dados. Porém, além da manipulação em si, eles também precisam oferecer meios de garantir integridade, consistência, recuperação de falhas, acesso simultâneo de múltiplos usuários e controle de concorrência, segurança, entre outros recursos. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

Um dos conceitos mais importantes na teoria de bancos de dados e, o mínimo exigido para que um SGBDR possa ser considerado utilizável comercialmente é conhecido como ACID (Atomicidade, Consistência, Isolamento e Durabilidade), termo muito utilizado para caracterizar transações. Para um melhor entendimento, segue abaixo as premissas de cada uma das características que formam a sigla do conceito:

- Atomicidade: Tem o sentido de indivisibilidade, garantindo que durante uma transação todas as alterações no banco de dados serão realizadas ou nenhuma delas será realizada. (ISSA, 2011 apud SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

  O exemplo mais conhecido e utilizado em atomicidade são os comandos *COMMIT* e *ROLLBACK*.
- Consistência: É a garantia de que o banco de dados sempre passará de um antigo estado consistente para um novo estado também consistente. (ISSA, 2011 apud SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

  Por exemplo, uma transação sempre terá que respeitar as integridades de chaves primárias e estrangeiras para ser finalizada com sucesso.
- Isolamento: Garante que durante a execução de várias transações concorrentes (acesso multiusuário) acessando um mesmo registro ou conjunto de registros, nenhuma delas irá interferir na execução da outra, podendo manipular os dados apenas após o término da primeira. (ISSA, 2011 apud SILBERSCHATZ; KORTH; SUDARSHAN, 2006)
- Durabilidade: É a capacidade do SGBD de manter o estado atual do banco de dados inalterado após o fim de uma transação, até que seja

> realizada uma nova transação que altere o seu estado novamente. (ISSA, 2011 apud SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

2.2.1 Modelo Relacional

O Modelo Relacional é utilizado por SGBDRs como forma de armazenamento e consiste basicamente na ideia da representação de dados em tabelas que se relacionam entre si. Cada tabela é formada por colunas que representam e definem o tipo de informação que será armazenada na mesma. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006, p. 5)

Os tipos mais conhecidos e que existem na maioria dos SGBDRs são: *CHAR*, *VARCHAR*, *INT*, *FLOAT*, *TIMESTAMP*, *BLOB*, entre outros.

Durante a criação da estrutura da tabela deve-se escolher uma ou mais colunas e defini-la(s) como chave(s) primária(s). As chaves primárias são colunas de uma tabela onde o valor de cada registro (tupla) é único, isto é, para que uma coluna possa ser definida como chave primária é necessário que os seus valores não se repitam em mais de uma tupla.

As tuplas são os registros do banco de dados e muitas vezes são chamadas também de linhas (*rows*). As tuplas são em si o conjunto de dados de uma tabela que representam uma informação no banco de dados.

Na Figura 1 abaixo, é possível observar um exemplo simples do modelo de dados relacional onde são descritas as colunas, tuplas e a chave primária de uma tabela onde são armazenados dados de endereços.

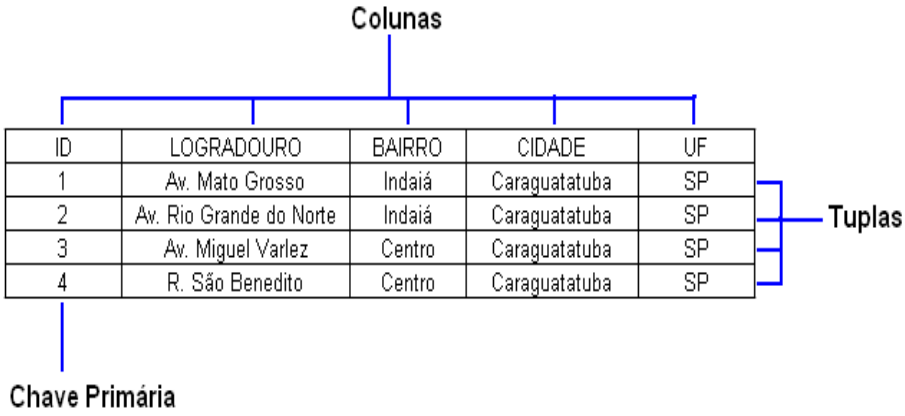


Figura 1: Exemplo da estrutura de uma tabela do modelo de dados relacional.

2.2.2 Tipos de Relacionamentos

Como citado anteriormente, as tabelas do modelo relacional podem se interligar por meio dos relacionamentos. Os relacionamentos são utilizados para evitar duplicidade de informações e diminuir o espaço de armazenamento em um banco de dados.

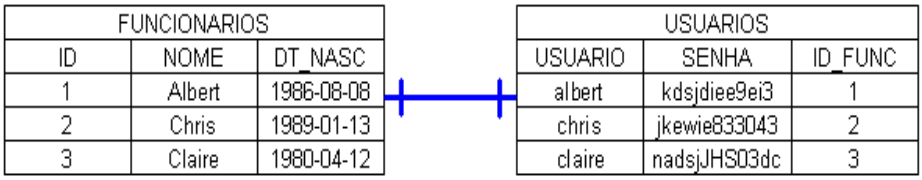
Tomando por base o exemplo da Figura 1 tem-se uma tabela de endereços, portanto, no caso de se construir uma segunda tabela que armazene informações sobre pessoas, não haverá necessidade de se criar colunas para armazenar dados de endereço para esta. Pois, basta um simples relacionamento entre ambas para que a nova tabela de pessoas passe a ter também dados da tabela de endereços.

Para que exista um relacionamento entre duas tabelas em um modelo relacional é necessária que em uma delas exista uma chave estrangeira. Define-se chave estrangeira por uma coluna de uma tabela que faz referência a uma chave primária de outra tabela.

Existem três tipos básicos de relacionamento entre tabelas em um banco de dados relacional, sendo eles:

- 1 : 1 (lê-se, um para um): Significa que cada tupla de uma tabela A pode fazer referência à apenas uma tupla de uma tabela B. Um exemplo deste relacionamento é um esboço de acesso a um sistema por funcionários de uma organização (um funcionário pode ser apenas um usuário no

sistema e um usuário pode ser apenas um funcionário), como pode ser observado na Figura 2.

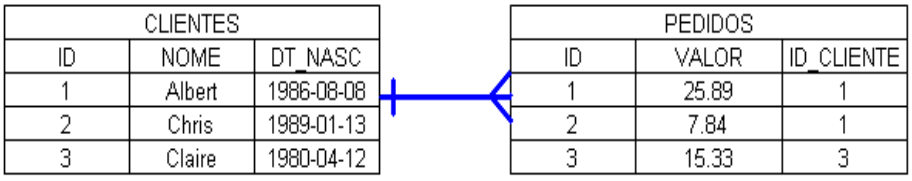


FUNCIONARIOS		
ID	NOME	DT_NASC
1	Albert	1986-08-08
2	Chris	1989-01-13
3	Claire	1980-04-12

USUARIOS		
USUARIO	SENHA	ID_FUNC
albert	kdsjdiee9ei3	1
chris	jkewie833043	2
claire	nadsjJHS03dc	3

Figura 2: Exemplo de um relacionamento 1:1.

- 1 : n (lê-se, um para muitos): Significa que cada tupla de uma tabela A pode fazer referência à várias tuplas de uma tabela B. Um exemplo deste relacionamento é um esboço entre pedidos e clientes (um cliente pode realizar vários pedidos, mas um pedido pode ser realizado por apenas um cliente), como pode ser observado na Figura 3.



CLIENTES		
ID	NOME	DT_NASC
1	Albert	1986-08-08
2	Chris	1989-01-13
3	Claire	1980-04-12

PEDIDOS		
ID	VALOR	ID_CLIENTE
1	25.89	1
2	7.84	1
3	15.33	3

Figura 3: Exemplo de um relacionamento 1:n.

- n : m (lê-se, muitos para muitos): Significa que várias tuplas de uma tabela A podem fazer referência à várias tuplas de uma tabela B. Na maioria dos SGBDRs é necessária a criação de uma terceira tabela que servirá para armazenar as chaves estrangeiras das primeiras. Um exemplo deste relacionamento é um esboço de uma venda (uma venda possui muitos itens e um item pode pertencer à mais de uma venda), como pode ser observado na Figura 4.

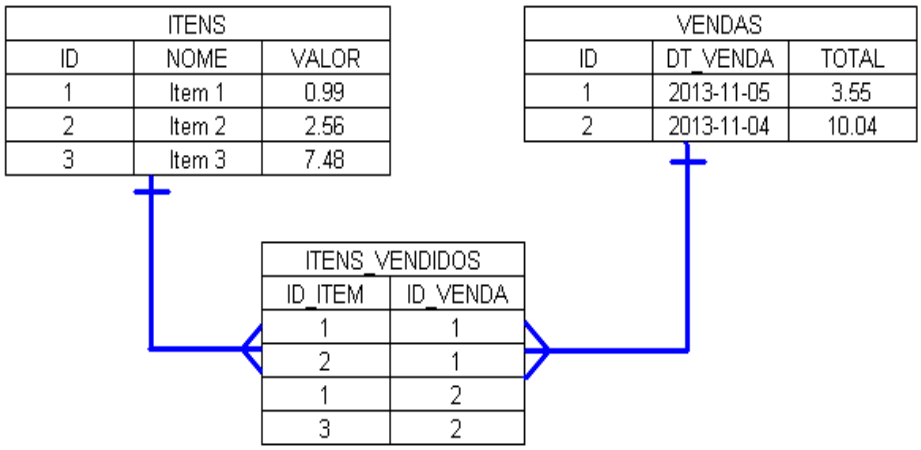


Figura 4: Exemplo de um relacionamento n:m.

2.2.3 SQL – *Structured Query Language* (Linguagem de Consulta Estruturada)

Como mencionado anteriormente, antes dos SGBDRs passarem a ter uso comercial em grande escala, cada programa que necessitasse armazenar dados de forma persistente precisava utilizar-se de arquivos em formato de texto ou binário, para garantir que as informações salvas durante o dia-a-dia não se perdessem e pudessem ser utilizadas posteriormente.

Contudo, cada sistema tinha uma maneira própria de manipular as inserções e recuperações das informações em cada arquivo, o que tornava quase inviável, por exemplo, que dois sistemas diferentes (desenvolvidos por empresas distintas) partilhassem de um mesmo arquivo de armazenamento. Pois, isso forçava com que ao menos uma das empresas tivesse um conhecimento muito familiar com o modo como a outra os utilizava para poder ter acesso às informações nele contidas.

Em meados de 1970 enquanto trabalhavam na IBM, Donald D. Chamberlin e Raymond F. Boyce desenvolveram uma linguagem de consulta que foi batizada de SEQUEL e fazia parte de um projeto chamado, projeto R. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006, p. 51)

Na mesma época, outra linguagem chamada de QUEL estava sendo utilizada pelo SGBD Ingres. A partir da junção destas duas linguagens surgiu mais tarde a Linguagem de Consulta Estruturada (SQL), que se tornaria a linguagem padrão de consulta em bancos de dados relacionais.

A linguagem SQL é utilizada em sistemas gerenciadores de banco de dados relacionais tanto para a modelagem do banco de dados (criação de tabelas e esquemas), quanto para a manipulação de dados (geração de relatórios, pesquisas com filtro, inserções, atualizações, exclusões, etc.). Ao contrário de outras linguagens de programação de uso geral, a SQL não existe fora do modelo relacional e não é possível criar aplicações somente com o uso dela.

Apesar de existir alinguagem SQL padrão (conhecida como SQL *Standard*) que foi publicada pela primeira vez em 1986 pela ANSI (sigla em inglês para, *American National Standards Institute*), cada SGBD tem uma linguagem SQL própria que diferem em detalhes de sintaxe umas das outras de acordo com o sistema em uso.

Devido a essas pequenas diferenças particulares diz-se que os SGBDRs atuais se utilizam de “dialetos” SQL. Neste trabalho será focado o uso do dialeto SQL do SGBD MSSQL Server 2012 também conhecida como *Transact*-SQL ou simplesmente T-SQL.

2.2.4 Subconjuntos da SQL

Dependendo da literatura, a linguagem SQL pode ser dividida em 3, 4 ou 5 subconjuntos de comandos principais dependendo das ações que eles exercem nos bancos de dados, sendo eles:

- DML - *Data Manipulation Language* (Linguagem de Manipulação de Dados): É o subconjunto da linguagem SQL que oferece comandos para a manipulação de dados, os comandos mais conhecidos são: *INSERT*, *UPDATE* e *DELETE*. Dependendo da literatura, o comando *SELECT* também pode ser considerado como parte integrante do subconjunto DML;

- DDL - *Data Definition Language* (Linguagem de Definição de Dados): É o subconjunto da linguagem SQL que oferece comandos para a definição das estruturas e relacionamentos dos dados, os comandos mais conhecidos são: *CREATE* e *DROP*. Dependendo do SGBD em uso também é possível utilizar o comando *ALTER*;
- DCL - *Data Control Language* (Linguagem de Controle de dados): É o subconjunto da linguagem SQL que oferece comandos para autorização de acesso a dados, os comandos mais conhecidos são: *GRANT* e *REVOKE*;
- DQL - *Data Query Language* (Linguagem de Consulta de Dados): Considerado por muitas literaturas o mais importante dos subconjuntos da linguagem SQL, este subconjunto é o responsável por realizar consultas nas bases de dados. O comando mais conhecido deste subconjunto é o *SELECT*, embora algumas literaturas o considerem como parte integrante da DML;
- DTL - *Data Transaction Language* (Linguagem de Transação de Dados): É o subconjunto da linguagem SQL que oferece comandos para o controle de transações, os comandos mais conhecidos são: *COMMIT* e *ROLLBACK*.

2.2.5 MSSQL Server

O SQL Server teve seu início em meados do ano de 1988 vindo de uma parceria entre as empresas Microsoft Corporation e Sybase. Contudo, em 1994 houve o rompimento da parceria entre ambas e desde então a Microsoft passou a manter as atualizações e manutenções do sistema.

Na atualidade, é um dos SGBDRs mais utilizados tanto em pequenas quanto em grandes aplicações que trabalham em ambientes operacionais Microsoft Windows, além de possuir uma vasta gama de ferramentas de gerenciamento e extração de relatórios empresariais, dependendo da sua edição. (http://db-engines.com/en/ranking, 2013)

2.2.6 Versões do SQL Server 2012

A seguir há uma breve descrição das versões oferecidas na edição 2012 do SQL Server:

- *Enterprise Edition*: É a versão mais completa, qual oferece todos os recursos disponíveis para os mais diversos fins dentro de uma empresa, recursos estes que vão desde ferramentas de gerenciamento até mesmo a ferramentas avançadas de BI, além de espelhamento assíncrono e segurança avançada;
- *Standard Edition*: É uma versão mais leve que oferece algumas ferramentas de BI mais básicas e não tem sistema de segurança tão avançado quanto a versão *Enterprise*. É recomendado para uso não crítico e o seu preço é relativamente mais barato;
- *Business Intelligence Edition*: É uma versão que oferece basicamente as ferramentas de BI presentes na versão *Enterprise*;
- *Developer Edition*: É uma versão idêntica ao *Enterprise Edition* para uso em desenvolvimento e testes. Possui um baixo valor e não pode ser utilizado em ambientes de produção;
- *Express Edition*: Esta versão do SQL Server é gratuita e pode ser obtida pelo site da Microsoft. É recomendada para pequenas aplicações e também para desenvolvedores que necessitam de uma versão do SQL Server com um mínimo de requisitos.

2.3 SISTEMA DE GERENCIAMENTO DE BANCOS DE DADOS NOSQL

O termo NoSQL foi criado por Carlo Strozzi em 1998, quando ele desenvolveu um sistema gerenciador de banco de dados que não possuía a linguagem SQL como padrão. Tecnicamente, este termo é inapropriado para o movimento qual ele representa, pois na verdade a principal diferença que estes gerenciadores trazem é a falta de relacionamentos (não se utilizam do modelo relacional).

Mas mesmo antes deste termo ser criado, mais precisamente desde a década de 60, já existiam bancos de dados não relacionais. Nesta época já se utilizavam os bancos de dados orientados a grafos, orientados a objetos, entre outros. Porém, recentemente novas abordagens foram criadas, como os bancos de dados orientados a coluna e os orientados a documentos.

Estas novas tecnologias surgiram para auxiliar em situações onde é difícil registrar a realidade por meio de estruturas rígidas como as tabelas, ou quando a representação utilizando o modelo relacional torna-se um empecilho em termos de desempenho.

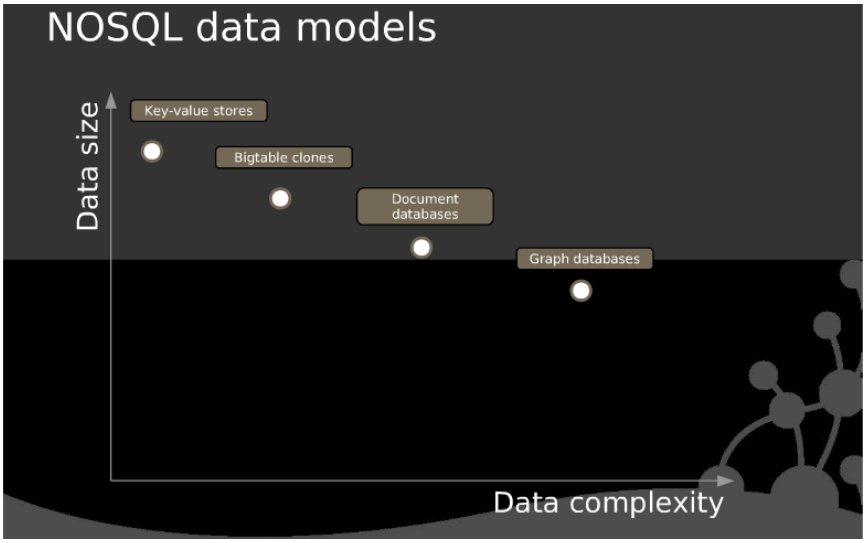


Figura 5 - Tabela de escolha de Banco de Dados com base na necessidade (Fonte: http://www.tomsitpro.com/articles/rdbms-sql-cassandra-dba-developer,2-547-2.html; 2013).

Como pode-se observar na Figura 5, acima, existem diferentes tipos de banco de dados NoSQL e, cada um é indicado para uma aplicação específica dependendo da quantidade de dados ou da complexidade dos dados. Os principais tipos de bancos de dados não relacionais são:

- Chave-Valor: Considerado como o NoSQL com maior poder de escalabilidade, ele é caracterizado por apresentar uma chave e um conjunto de valores representados por esta chave;
- Orientados a Coluna: Caracterizam-se por armazenar os dados por atributos e não por tuplas, otimizando assim o processo de busca de

alguma informação, pois só serão carregadas na memória RAM as colunas que estão sendo utilizadas;

- Orientados a Grafos: Diferentemente de outros bancos de dados, estes não armazenam registros, mas sim objetos, sendo estes ligados de forma semelhante a uma árvore de decisão;
- Orientados a Documentos: Sua maior característica é não possuir uma estrutura fixa de armazenamento, guardando coleções de documentos JSON ou XML, sendo possível uma mesma coleção conter vários documentos diferentes.

Os bancos de dados não relacionais caracterizam-se por respeitarem as propriedades BASE (*Basically Available, Soft state, Eventual consistency*), onde, o sistema não é consistente todo tempo, mas sim eventualmente, dando uma maior ênfase à disponibilidade e escalabilidade. Esta característica serve para firmar que o NoSQL não foi criado para substituir os bancos de dados relacionais, mas sim para serem utilizados em determinadas situações.

Outra característica marcante do NoSQL é a possibilidade de escalonamento horizontal, possibilitando que as informações fiquem distribuídas em várias máquinas comuns ao invés de usar grandes servidores como é feito no escalonamento vertical. A técnica utilizada para este procedimento chama-se *MapReduce*, que segundo Schneider (2012), “É construído sobre o conceito comprovado de dividir e conquistar: é muito mais rápido quebrar uma enorme tarefa em pequenos pedaços e processá-los em paralelo.”

O *MapReduce* trabalha dividindo o processamento em duas partes, o *Map* e o *Reduce*. O *Map* quebra as informações em pequenos nós que são distribuídos na rede, já o *Reduce* fica responsável por agregar esses nós gerados pelo *Map*, possibilitando assim a obtenção da informação completa, seguindo uma ideia de árvore de decisão. Um exemplo destes procedimentos pode ser observado nas figuras 6 e 7, abaixo.

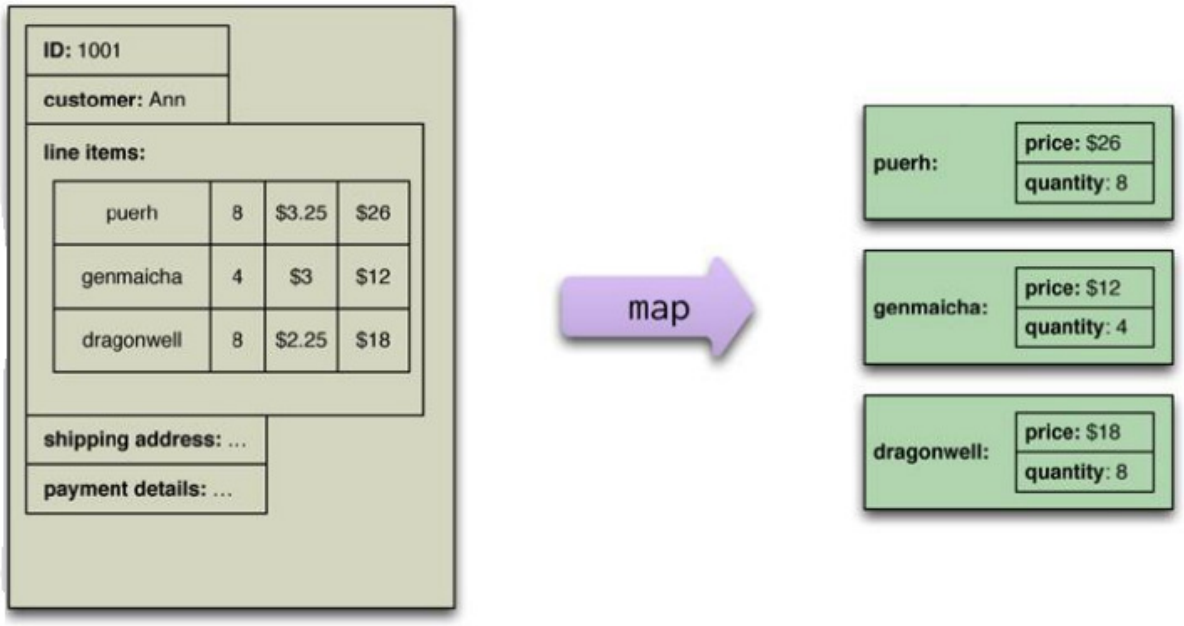


Figura 6: Modelo de Map (Fonte: http://www.tomsitpro.com/articles/rdbms-sql-cassandra-dba-developer,2-547-2.html; 2013).

Figura 7: Modelo de Reduce (Fonte: http://www.tomsitpro.com/articles/rdbms-sql-cassandra-dba-developer,2-547-2.html; 2013).

2.3.1 JSON – *JavaScript Object Notation*

Java Script Object Notation (JSON) é considerado um subconjunto da linguagem de programação JavaScript e, um formato “leve” de intercâmbio de dados de fácil leitura e escrita tanto para humanos, quanto para máquinas. (http://json.org; 2013)

Surgiu como uma alternativa à “pesada” Linguagem de Marcação Estendida (XML) que também é utilizada como formato de intercâmbio de dados. A estrutura simplificada do JSON permite que se passe um volume maior de informações entre o emissor e o receptor, com um gasto mais baixo de recursos.

A sua “leveza” em relação ao XML está no fato do JSON descrever estados de objetos através das combinações de chave-valor, que ocupam muito menos espaço do que as tradicionais TAGs utilizadas no XML. Uma comparação entre as sintaxes das duas linguagens mostrando a sensação de leveza superior do JSON pode ser observada na Figura 8.

Formato XML

```
<?xml version="1.0" ?>
<pessoa>
    <nome>
        Alexia Ashford
    </nome>
    <idade>
        30
    </idade>
</pessoa>
```

Formato JSON

```
pessoa = {
    "nome": "Alexia Ashford",
    "idade": 30
}
```

Figura 8: Comparação entre os formatos de anotação XML e JSON.

2.3.2 MongoDB

Segundo Banker (2011), em seu livro MongoDB in Action:

> *"MongoDB é um sistema de gerenciamento de banco de dados projetado para aplicações web e infra-estrutura de internet. As estratégias de modelo de dados e persistência são construídas para uma alta vazão de leitura e escrita e, uma capacidade de escalar facilmente com failover automático. Se uma aplicação requer apenas um nó de banco de dados ou dezenas deles, em ambos os casos, MongoDB pode fornecer desempenho surpreendentemente bom. Se você tem enferentado dificuldades escalando bancos de dados relacionais, esta pode ser uma ótima notícia. Mas nem todo mundo precisa operar em escala. Talvez tudo que você sempre precisou, é de um único servidor de banco de dados."*

O MongoDB foi criado pela empresa 10gen no ano de 2007 e seu nome origina-se da palavra *humongous* que, em português significa gigantesco. É um SGBDde código aberto orientado a documentos que possui versões de 32 bits e 64 bits, podendo ser feito o download da versão em pacotes ou na versão binária. Foi escrito em C++ e executa comandos utilizando a linguagem JavaScript, que é uma linguagem de programação interpretada muito difundida no ambiente WEB, além de usar um formato leve de intercâmbio de dados conhecido como JSON. (BANKER, 2011, p. 5)

Todo documento persistido pelo usuário no MongoDB é no formato JSON, porém, internamente é utilizado o BSON que é um formato binário do JSON, sendo esse processo de transformação realizado automaticamente, como é descrito na Figura 9.

Porém, para realizar esta transformação o MongoDB acaba utilizando-se de uma grande quantidade de memória RAM, uma vez que o seu algoritmo sempre aloca o dobro de memória utilizada. Apesar deste comprometimento do espaço de memória, este processo faz com que os dados fiquem armazenados de forma sequencial, auxiliando assim, em pesquisas.

O BSON possui um limite de tamanho de 16MB e para armazenamento de arquivos maiores é utilizado o GridFS, o qual divide o arquivo em diversos pedaços limitando-os em 256k e os armazena em uma coleção. Para que os mesmos sejam montados novamente, é criada outra coleção onde são guardadas a ordem e a posição de cada pedaço do arquivo.

Além de servir para armazenar arquivos grandes, o GridFS também serve para armazenar arquivos que não precisam ser totalmente carregados na memória para que possam ser acessados.

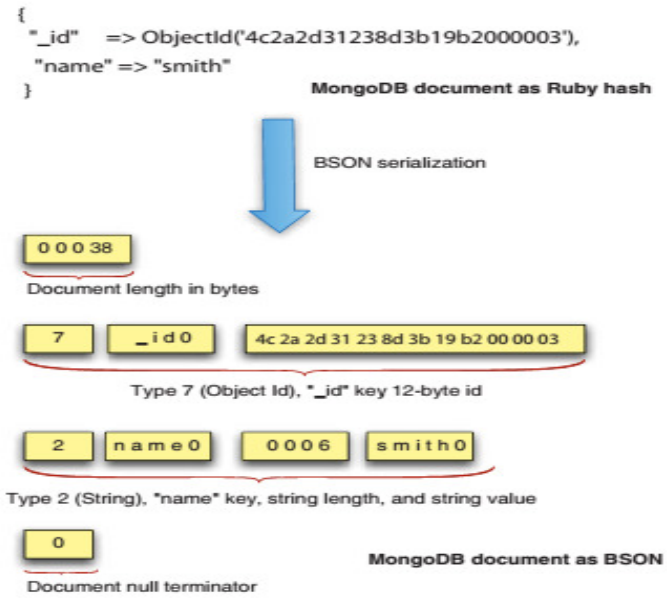


Figura 9: Conversão de JSON para BSON (Fonte: http://www.tomsitpro.com/articles/rdbms-sql-cassandra-dba-developer,2-547-2.html; 2013).

Como a maioria dos bancos de dados NoSQL, o MongoDB também foi projetado para permitir o escalonamento horizontal e, para isso existe o processo de *sharding* nativo, onde as informações são divididas em pequenos pedaços chamados de *shards*. Esse procedimento é feito por coleção e para que ele ocorra, a quantidade de dados deve ser grande. (CHODOROW; DIROLF, 2013, p. 143-145)

Na Tabela 1 é apresentada uma comparação entre as terminologias utilizadas em banco de dados relacionais e as utilizadas pelo MongoDB, assim é possível ter uma ideia melhor acerca dos termos técnicos entre ambos.

Relacional	Mongo
Database	Database
Table	Collection
Tupla	Document
Query	Json
Index	Index
Partition	Shard

Tabela 1: Comparação de terminologias relacional e NoSQL.

3 COMPARAÇÕES ENTRE MS SQL SERVER E MONGODB

3.1 COMPARAÇÃO DE SINTAXE

Como já foi dito anteriormente o MS SQL Server 2012 utiliza como padrão a linguagem SQL e, o MongoDB utiliza a linguagem JavaScript. A seguir será demonstrada uma comparação entre os principais comandos dos dois gerenciadores.

3.1.1 Criando um banco de dados

Para se criar um banco de dados no MS SQL Server 2012 são necessários três passos básicos:

- Criar um banco de dados:

  CREATE DATABASE <nome_do_banco_de_dados>;
- Selecionar o banco de dados criado:

  USE loja;
- Criar as tabelas:

  CREATE TABLE [dbo].[itens](

  <campo><tipo><propriedades>,

  <campo><tipo><propriedades>,

  <campo><tipo><propriedades>,

  PRIMARY KEY(<campo>)

  );

No MongoDB basta executar o comando de seleção do banco de dados que o próprio gerenciador se encarregará de criá-lo quando o primeiro documento for

inserido em alguma coleção. Este comando também é utilizado para selecionar um banco de dados já existente.

use<nome_do_banco_de_dados>;

3.1.2 Inserção de dados

Este comando é utilizado para inserir novos dados em um banco. No MS SQL Server 2012 é preciso respeitar a estrutura das tabelas criadas, bem como o tipo de cada atributo, garantindo assim que todos os dados recebidos nesta tabela respeitem os padrões pré-estabelecidos. Enquanto que, no MongoDB basta inserir as informações com a estrutura desejada e sem nenhuma preocupação com tipos de dados, podendo ainda, cada documento da coleção conter uma estrutura diferente.

A Tabela 2, abaixo, mostra a diferença de sintaxe entre os SGBDs em uma operação de inserção de dados.

MS SQL Server 2012	MongoDB
INSERT INTO<nome_tabela> (<campo 1>, ...<campo n>) VALUES (<valor 1>, ... <valor n>);	db.<coleção>.insert(<novo documento>)

Tabela 2: Comparação de comandos de inserção de dados.

3.1.3 Seleção de dados

Para buscar informações já cadastradas em um banco de dados é utilizado o comando de seleção. Este comando possui diversos recursos, sendo possível resgatar os dados de diferentes formas. A título de exemplo, a Tabela 3 mostra uma seleção de todas as vendas cadastradas em um banco de dados.

MS SQL Server 2012	MongoDB
SELECT <campos> FROM <nome_tabela>;	db.<coleção>.find()

Tabela 3: Comparação de comandos de seleção de dados.

Em ambos gerenciadores é possível selecionar informações de forma mais detalhada, impondo condições e/ou projetando apenas alguns campos específicos, como pode ser observado na Tabela 4, a seguir.

MS SQL Server 2012	MongoDB
SELECT <campos> FROM <nome_tabela> WHERE <condição>;	db.<coleção>.find(<campos>, <condição>)

Tabela 4: Comparação de comandos de seleção avançada de dados.

3.1.4 Atualização de dados

Existem momentos em que algum dado inserido em um banco de dados necessita de alguma alteração, por exemplo, o preço de um produto que por diversas vezes terá seu valor alterado ou as taxas atualizadas. A Tabela 5 exemplifica uma alteração de valores de um registro (SQL Server) e de um documento (MongoDB).

MS SQL Server 2012	MongoDB
UPDATE <nome_tabela> SET <campo 1> = <novo valor>, ..<campo n> = <valor n> WHERE <condição>;	db.<coleção>.update(<condição>, <novos valores>, <opções>)

Tabela 5: Comparação de comandos de atualização de dados.

No MongoDB existem duas cláusulas obrigatórias na hora de se fazer uma atualização de dados, sendo elas, o *upsert* e o *multi*. As duas existem mesmo se não forem citadas nos comandos e por padrão possuem o valor *false*.

O *upsert* quando definido com *true*, cria um novo documento quando nenhum outro corresponde com os critérios da consulta, quando definido como *false* faz com que somente haja a atualização dos documentos já inseridos.

O *multi* determina se apenas o primeiro documento encontrado na seleção será atualizado (*false*) ou, se todos os documentos da coleção serão alterados (*true*).

3.1.5 Remoção de dados

Este comando serve para remover dados cadastrados em um banco de dados. Um exemplo da operação de remoção pode ser observado na Tabela 6.

MS SQL Server 2012	MongoDB
DELETE FROM <tabela>;	db.<coleção>.remove()
DELETE FROM <tabela> WHERE <condição>;	db.<coleção>.remove(<condição>)

Tabela 6: Comparação de comandos de remoção de dados.

4 INSTALAÇÃO DE FERRAMENTAS E TESTES DOS SGBDS

Este capítulo é dedicado à demonstração dos passos que foram realizados durante a instalação e configuração das ferramentas necessárias para a realização dos testes. Mostrando que o cenário montado não tendeu a favorecer nenhum dos SGBDs com a realização de configurações mais avançadas para prover, por exemplo, mais velocidade. Isso é a garantia de que ambos os sistemas foram utilizados da forma mais crua possível.

4.1 INSTALAÇÃO DE FERRAMENTAS

Nesta seção serão abordados os passos que foram executados para a instalação dos recursos e ferramentas para a preparação do ambiente para os testes dos SGBDs. Todos os softwares foram instalados em um microcomputador pertencente ao Instituto Federal de Educação, Ciência e Tecnologia do Estado de São Paulo – Campus Caraguatatuba, com as seguintes configurações:

- Processador Intel Core i5-3470s de 2,9GHz;
- 2 GB de memória RAM;
- 500 GB de HD e
- Sistema Operacional Microsoft Windows 7 Professional.

4.1.1 Instalação do SQL Server 2012

A instalação do SQL Server 2012 não é difícil, mas sim, um tanto trabalhosa devido as suas longas etapas de verificações do sistema em busca de compatibilidade de recursos, o que a torna complexa e exige bastante atenção nas mensagens que o instalador mostra durante o processo.

Com o DVD de instalação do SQL Server 2012 na bandeja basta clicar no arquivo executável que inicia a instalação e será aberta a tela inicial mostrada na Figura 10. Nesta tela é necessário clicar no *link* “Instalação” no canto superior direito

e depois em “Nova instalação autônoma do SQL Server ou adicionar recursos a uma instalação existente” encontrado na parte superior do lado direito.

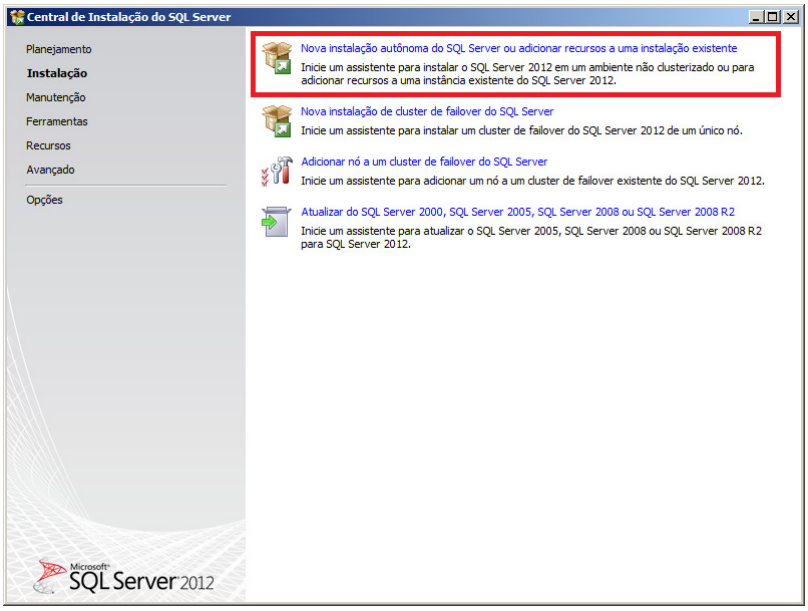


Figura 10: Início da instalação do SQL Server 2012.

As telas que se seguem são passos de verificação das condições do sistema operacional em busca de possíveis problemas ou, falta de recursos tanto de *hardware* quanto de *software* que podem vir a comprometer o processo de instalação no futuro. Após estes passos, a primeira tela da instalação do servidor do SQL Server pode ser vista na Figura 11.

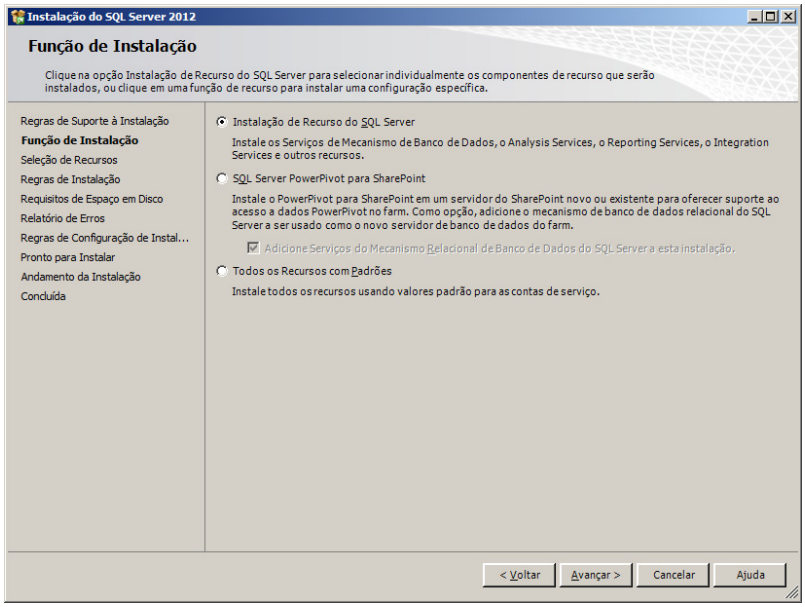


Figura 11: Tela inicial do instalador do servidor do SQL Server 2012.

Ainda na tela da Figura 11, é selecionada a opção “Instalação de Recurso do SQL Server” e clicado em “Avançar” para chegar à próxima tela onde haverá a possibilidade de escolha dos recursos a serem instalados no servidor, assim como os diretórios padrões de instalação, como pode ser visto na Figura 12, abaixo.

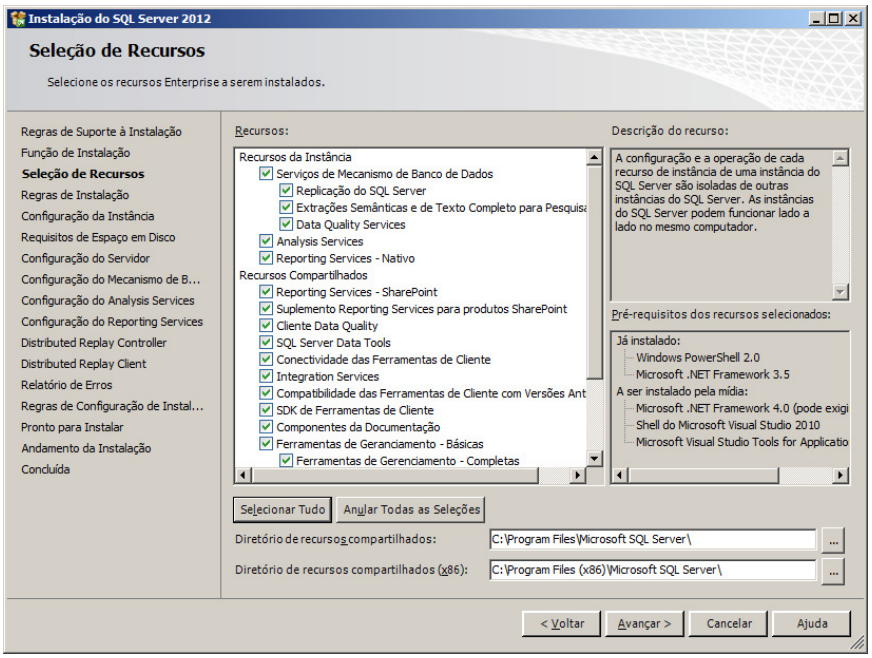


Figura 12: Tela de seleção dos recursos à serem instalados.

Para este trabalho foram selecionadas todas as opções de recursos disponíveis, mantidos os diretórios padrões dos recursos compartilhados tanto de 64 quanto de 32 *bits* e clicado em “Avançar”. Após outra verificação para identificar se o sistema operacional atende aos requisitos para a instalação dos componentes selecionados, é mostrada a tela de configuração da instância do servidor, onde também foram mantidos os valores padrão, como pode ser observado na Figura 13.

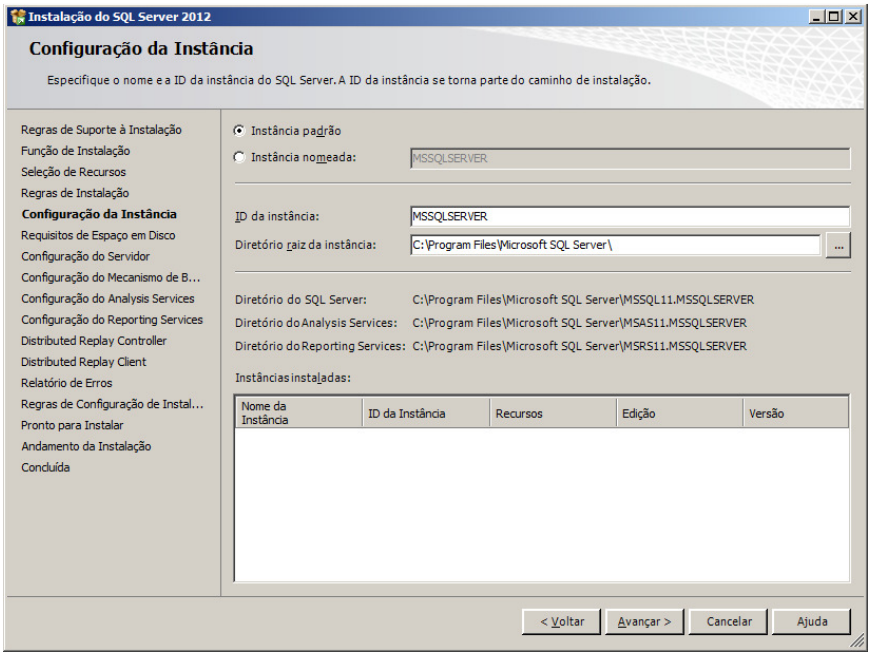


Figura 13: Tela de configuração da instância do SQL Server.

Ao ser clicado o botão “Avançar”, será apresentada uma tela com o resumo do espaço em disco que será utilizado pelo servidor e, na tela seguinte é possível escolher o tipo de inicialização de cada serviço. Ao clicar em Avançar será apresentada a tela de “Configuração do Mecanismo de Banco de Dados”, onde é possível escolher o modo de autenticação, conforme exibido na Figura 14.

Neste caso foi escolhido o modo misto que permite autenticação no banco de dados usando usuários criados no SGBD e/ou usuários do Windows. Foi adicionada senha ao usuário administrador do SQL Server (usuário chamado “sa”) e, adicionadas contas de usuário do próprio sistema operacional Windows para que as mesmas possam ter permissões de fazer *logon* no SQL Server e administrar as bases de dados.

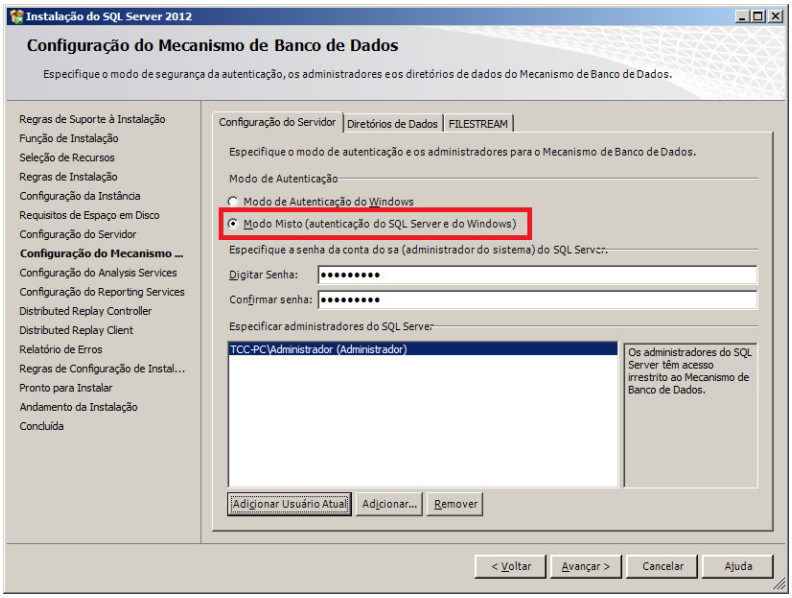


Figura 14: Configuração do mecanismo de banco de dados.

As telas que se seguem após a mostrada na Figura 14, são telas de configuração dos outros recursos que serão instalados, por exemplo, ferramentas de relatórios. Em todas elas foram mantidos os valores padrão das configurações e clicado no botão “Avançar”.Ao fim, se tudo estiver certo é possível ver a tela de confirmação de sucesso da instalação dos recursos, conforme pode ser observado na Figura 15.

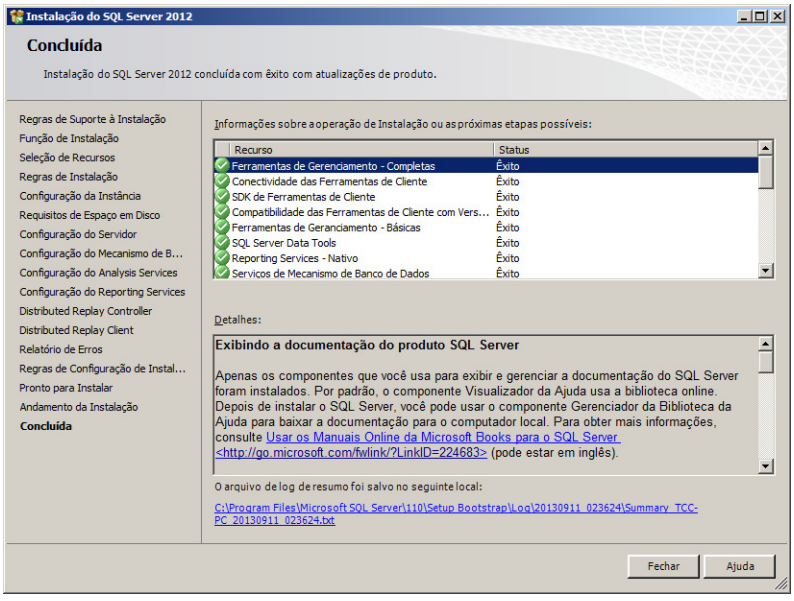


Figura 15: Conclusão da instalação com sucesso.

4.1.2 Instalação do MongoDB

A instalação de um servidor MongoDB é relativamente simples se comparado ao MS SQL Server 2012. Basta fazer o *download* do pacote de acordo com a plataforma do computador hospedeiro utilizando o *link* http://www.mongodb.org/downloads e descompactar o conteúdo do arquivo.

No caso de computadores que utilizam o sistema operacional Microsoft Windows, é aconselhável descompactar o conteúdo do arquivo no disco local C: e alterar o nome do diretório para mongodb, como no exemplo que pode ser visto na Figura 16.

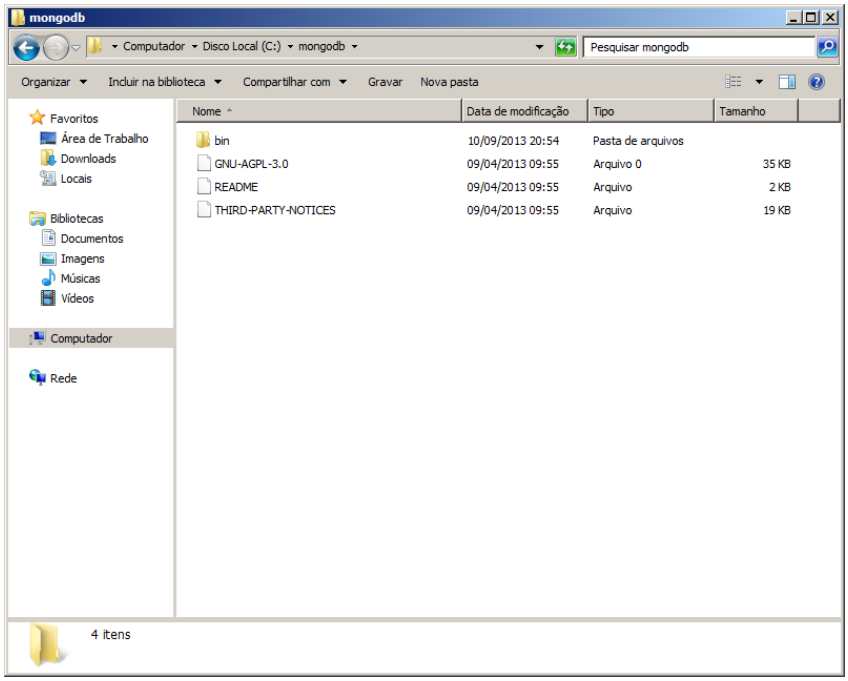


Figura 16: MongoDB descompactado.

Após este procedimento, se faz necessária a criação de um diretório que servirá de destino onde o MongoDB possa criar e gerenciar os bancos de dados. Para a realização deste trabalho foi criado um diretório chamado “data” no disco local C: e um subdiretório chamado “db” dentro do recém-criado “data”, tendo então no final o seguinte caminho: C:\data\db.

Após a criação do diretório de dados o MongoDB está pronto para ser configurado. Para isso é necessário abrir um *Prompt de Comando* e navegar até o

diretório C:\mongodb\bin onde se encontra o arquivo executável que inicia o servidor do MongoDB, chamado mongod.exe, e executá-lo passando como parâmetro o caminho onde será o diretório de dados, no caso C:\data\db, através da opção "--dbpath=" como pode ser visto na Figura 17.

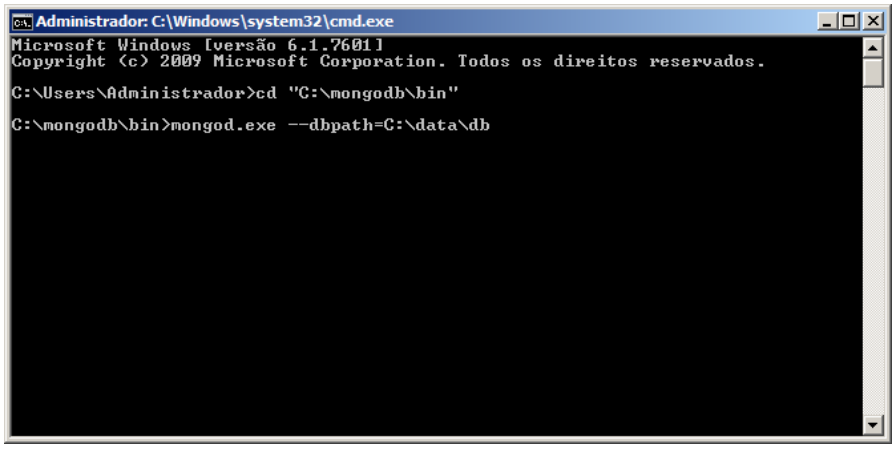


Figura 17: Inicialização do servidor MongoDB.

Caso o *firewall* do Windows esteja habilitado, será aberta uma janela *pop-up* pedindo o desbloqueio do programa mongod.exe para que o mesmo tenha acesso à rede (Figura 18).

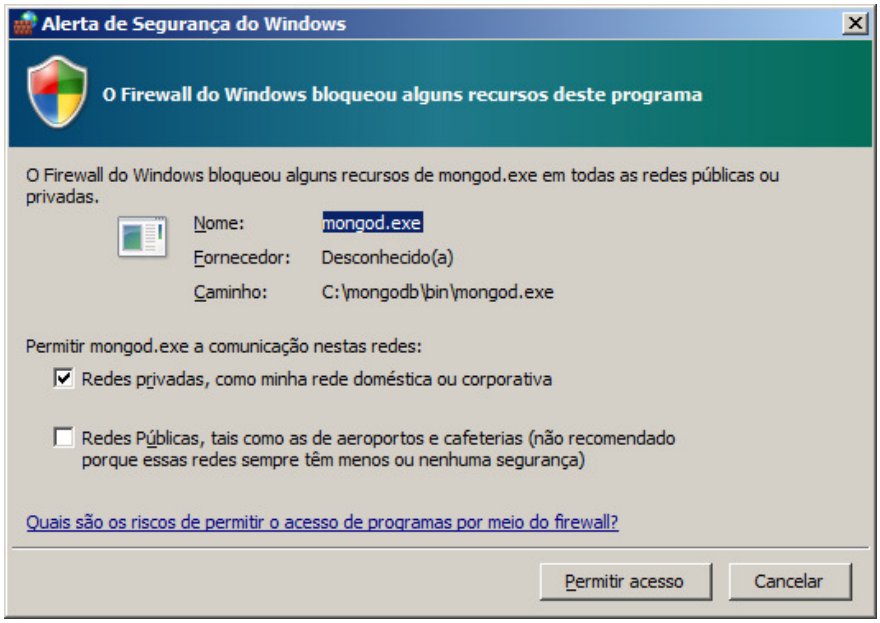


Figura 18: Permitir acesso do programa mongod.exe à rede.

Independente da janela do firewall do Windows aparecer ou não, caso tudo esteja correto, é possível ver o servidor do MongoDB executando e esperando por conexões na porta 28017 como pode ser visto na Figura 19.

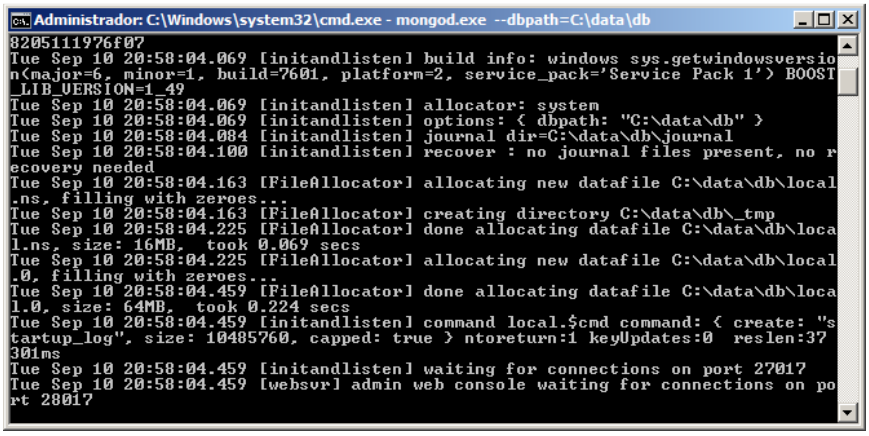


Figura 19: Servidor MongoDB iniciado e esperando por conexões.

Também é possível verificar que foram criados os primeiros arquivos de controle dos bancos de dados no diretório de dados que foi configurado como local de trabalho do mongod.exe, neste caso: C:\data\db (Figura 20).

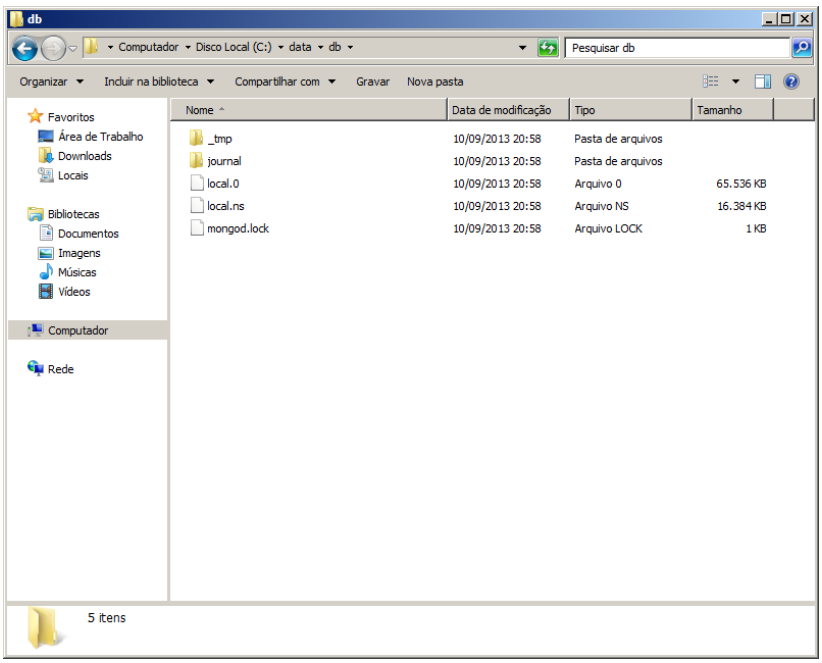


Figura 20: Primeiros arquivos de controle dos bancos de dados.

4.1.3 Instalação do Apache JMeter

O JMeter é uma ferramenta desenvolvida pela Apache Software Foundation e que faz parte do projeto Jakarta. O principal objetivo desta ferramenta é propiciar testes de carga e *stress* em sistemas computacionais, bancos de dados, etc. (SANTOS; NETO, 2008)

Para isso, o JMeter conta com uma grande gama de ferramentas disponíveis, onde, as utilizadas neste trabalho foram:

- Controle de *Threads* (*Thread Groups*): são utilizadas para simular quantos "usuários" acessarão simultaneamente o sistema sob teste, e quantas vezes cada um destes "usuários" o acessará;
- *Listeners*: traduzindo do inglês ao pé da letra, os "ouvintes" são variadas formas de ferramentas que recebem o que está acontecendo durante um plano de testes e, tem a finalidade de mostrar o resultado obtido através de gráficos, árvores de resultados, tabelas, etc.;
- Elementos de Configuração: pode-se compará-los às variáveis de ambiente, que têm a finalidade de armazenar as configurações necessárias para a execução dos testes;
- Testadores: estes são os elementos responsáveis por realizarem os testes utilizando-se dos elementos de configuração e das *threads* de usuários, além de outros recursos inclusos no plano de teste.

A instalação da ferramenta de testes JMeter é relativamente simples. Porém, para que a mesma tenha a possibilidade de realizar testes em bancos de dados, se faz necessária a instalação de *plugins* adicionais que, dependendo dos quais, não são encontrados no mesmo lugar onde a ferramenta é obtida. Para dar início a instalação e configuração do JMeter é necessária a escolha de um dos pacotes compactados disponíveis para download através do endereço http://jmeter.apache.org/download_jmeter.cgi.

Após a seleção e download do pacote mais apropriado ao sistema operacional em uso, basta descompactar o conteúdo do arquivo em um diretório qualquer - é aconselhável que se descompacte no disco local C:, quando se faz uso de sistemas operacionais Microsoft Windows - como pode ser visto na figura 21.

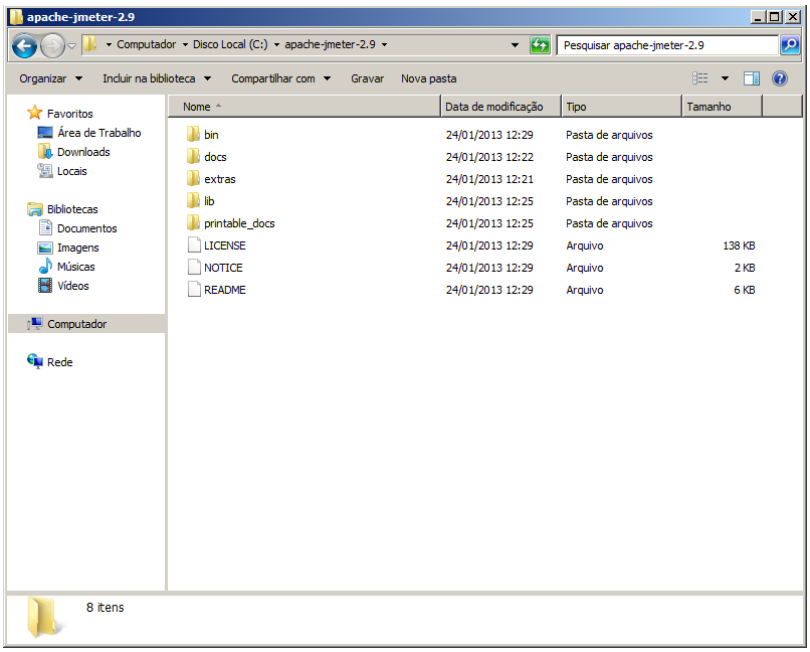


Figura 21: Estrutura de diretórios do JMeter após este ser descompactado.

Para que seja possível a comunicação entre o JMeter e o SQL Server 2012 é necessário obter o *driver* JDBC para SQL Server pelo endereço http://www.microsoft.com/pt-br/download/details.aspx?id=11774, e que os arquivos “sqljdbc.jar” e/ou “sqljdbc4.jar” que se encontram dentro do arquivo obtido sejam adicionados dentro do diretório “lib” da instalação do JMeter, como pode ser visto na Figura 22.

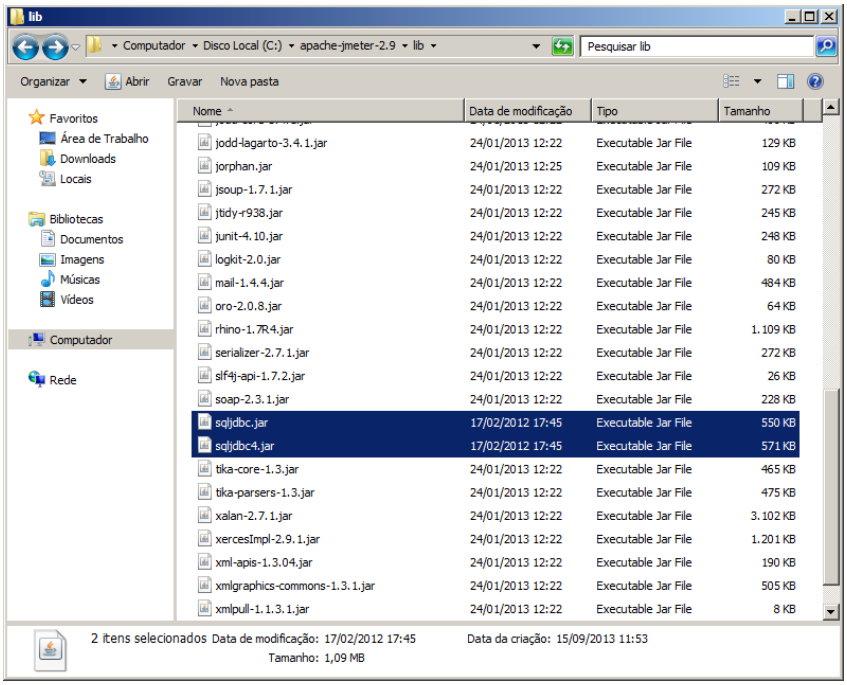


Figura 22: Instalação do driver JDBC para SQL Server.

Para que a comunicação entre o JMeter e o MongoDB seja possível, é necessária a obtenção de dois componentes encontrados em lugares diferentes na WEB. O primeiro deles é o *driver* de conexão JDBC para MongoDB, que tem a tarefa de realizar conexões entre o JMeter e o MongoDB e pode ser obtido através do endereço http://central.maven.org/maven2/org/mongodb/mongo-java-driver/2.4/.

O segundo componente é um *plugin* para o JMeter que permite a configuração de conexões com o MongoDB e a execução de requisições utilizando-se do *driver* JDBC, trata-se do *plugin* MongoMeter que está disponível para *download* no endereço https://github.com/JanPaulEttles/mongometer.

Após a obtenção de ambos os componentes, basta adicioná-los no diretório “lib\ext” do diretório de instalação do JMeter, como pode ser visto nas figuras 23 e 24.

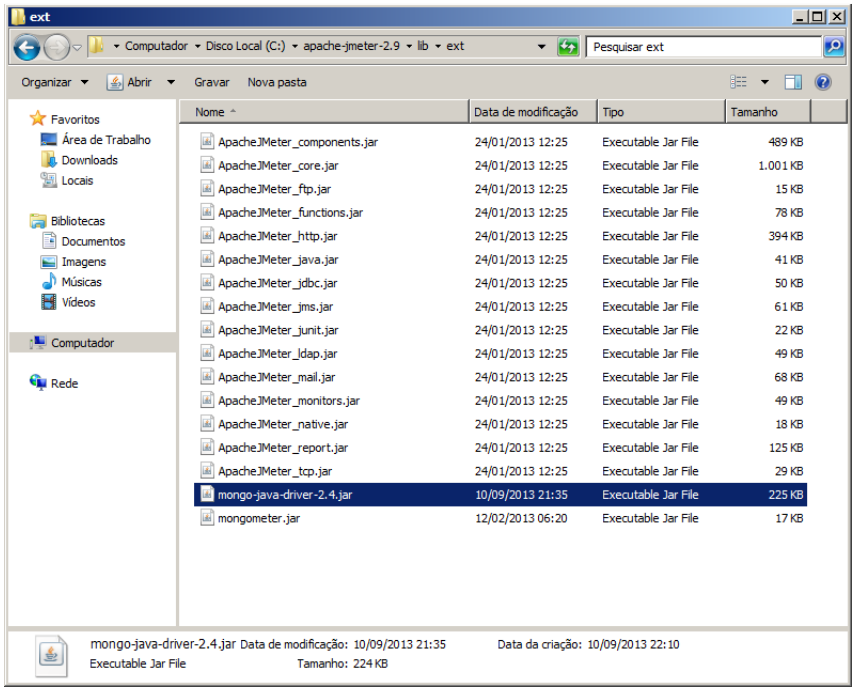


Figura 23: Instalação do driver JDBC para MongoDB.

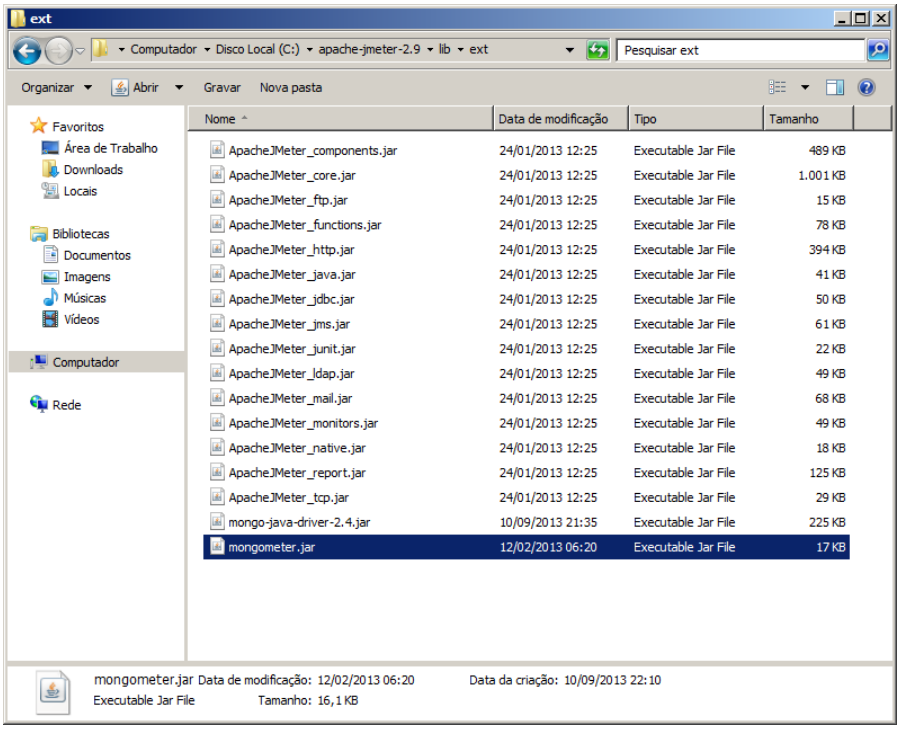


Figura 24: Instalação do plugin MongoMeter.

Uma vez seguidos os passos acima (mas não necessários), pode-se navegar pela árvore de diretórios da instalação do JMeter até o diretório “bin” e executar o arquivo “jmeter.bat” para que possa ser vista a sua tela inicial, como é mostrado na Figura 25.

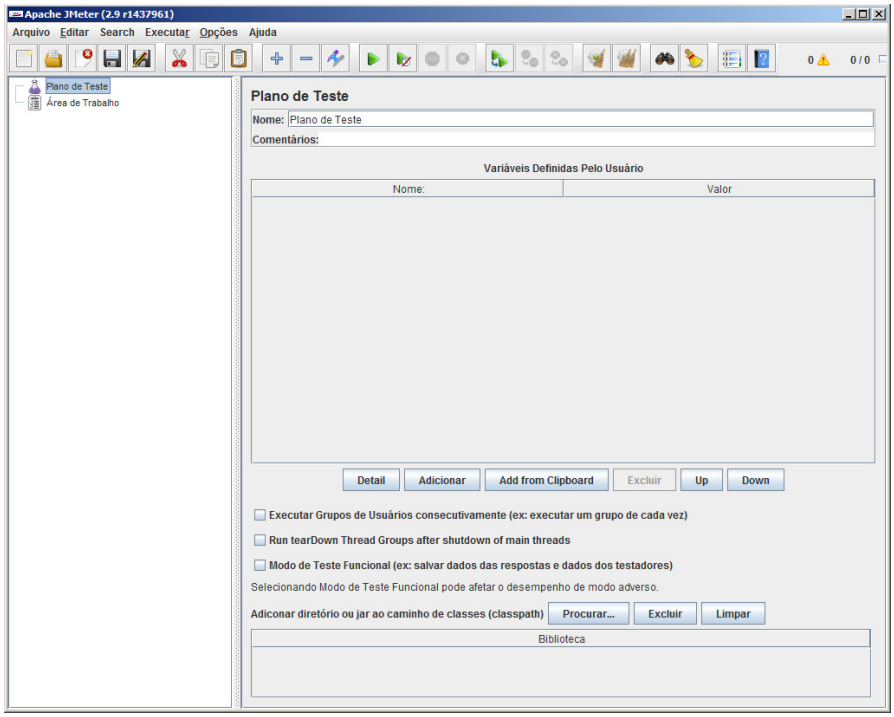


Figura 25: Tela inicial da ferramenta de testes de software Apache Jmeter.

4.2 TESTES DOS SGBDS

Para os testes, os dois SGBDs foram instalados e não passaram por nenhuma configuração mais avançada além da configuração inicial de instalação, uma vez que, o objetivo deste trabalho é verificar o desempenho de cada um deles em um cenário o mais simples possível.

Após a instalação dos recursos e ferramentas citados anteriormente, foi dado início aos testes em ambos os SGBDs utilizando-se de planos de testes montados e executados no Apache JMeter. Para a obtenção dos resultados foram utilizados três *listeners* do JMeter que mostram informações no formato de tabelas, árvores e tabelas agregadas.

No SQL Server 2012 foram criadas *Stored Procedures* que utilizam transações para executar as operações ao invés de *queries* simples executadas separadamente.

Foram testadas as quatro principais operações dos SGBDs (inserções, atualizações, remoções e seleções) utilizando-se somente de valores numéricos,

valores de data e texto com a codificação padrão de cada SGBD. A bateria de testes foi dividida em dois tipos sendo eles, testes de carga (realização de operações de forma sequencial e não concorrente) e testes de *stress* (realização de operações de forma concorrente).

Durante os testes dos SGBDs foram avaliados cinco requisitos, sendo eles descritos a seguir:

- Consumo de memória RAM: O consumo de memória RAM foi medido em MegaBytes (MB);
- Consumo de CPU: O consumo de processamento de CPU foi medido em porcentagem (%) de uso, tendo como base o total de processamento disponível no microcomputador;
- Taxa de erro: A taxa de erro foi medida em porcentagem (%) e se refere à quantidade de execuções que não foram concluídas com sucesso durante os testes;
- *Throughput*: É a quantidade de dados enviada em determindado espaço de tempo, que no caso do JMeter, é medida em KiloBytes por segundo (KB/s);
- Vazão: É a quantidade de comandos executados por segundo, que neste caso será demonstrado como N/s, onde: N = Número de execuções e, s = segundos.

Para se chegar a uma conclusão de qual SGBD tem aproveitamento melhor diante de outro, é necessário cruzar estas cinco informações de forma a avaliar o desempenho de cada um como um todo. Assim sendo, um SGBD que possui uma taxa de vazão maior não é necessariamente mais rápido que o outro se, este também não obtiver uma taxa de erro menor ou igual que a do concorrente.

No caso de se comparar a vazão em conjunto com o *throughput* de um SGBD, tem melhor desempenho aquele que consegue uma maior taxa de vazão com uma menor taxa de *throughput*. Por exemplo, tomando como base uma operação de recuperação de informações (*SELECT*), o SGBD que conseguir recuparar a mesma informação com uma maior vazão e um menor *throughput*, é o que tem um melhor desempenho.

Já os resultados destas duas comparações precisam ser comparados com o consumo de memória RAM e CPU de cada um, a fim de verificar qual dos dois gastou mais recursos do sistema para completar as requisições.

4.2.1 Testes de Carga

Os testes de carga foram divididos em três etapas de: 1.000 execuções, 10.000 execuções e 100.000 execuções onde, em cada uma delas foram executadas as operações básicas de manipulação de dados (*INSERT*, *UPDATE*, *DELETE* e *SELECT*).

O JMeter foi configurado para que cada etapa fosse executada de forma sequencial e não-concorrente, como descrito a seguir:

- Na etapa de 1.000 execuções foi simulado um único usuário realizando 1.000 transações;
- Na etapa de 10.000 execuções foi simulado um único usuário realizando 10.000 transações;
- Na etapa de 100.000 execuções foi simulado um único usuário realizando 100.000 transações.

4.2.2 Testes de *Stress*

Os testes de *stress* foram divididos em três etapas de: 1.000 execuções, 10.000 execuções e 100.000 execuções onde, em cada uma delas foram executadas as operações básicas de manipulação de dados (*INSERT*, *UPDATE*, *DELETE* e *SELECT*).

O JMeter foi configurado para que cada etapa fosse executada de forma concorrente, como descrito a seguir:

- Na etapa de 1.000 execuções foram simulados 1.000 usuários, cada um realizando 1 transação;

- Na etapa de 10.000 execuções foram simulados 1.000 usuários, cada um realizando 10 transações;
- Na etapa de 100.000 execuções foram simulados 1.000 usuários, cada um realizando 100 transações.

4.3 RESULTADOS

A seguir serão apresentados gráficos e tabelas com os resultados obtidos para cada operação realizada em cada estágio e cenário diferente.

4.3.1 Testes de Carga

Nos resultados dos testes da operação de *INSERT* executada 1.000 vezes por 1 usuário (Tabela 7 e Figura 26, abaixo) é possível notar um maior *throughput*, consumo de memória e processamento por parte do SQL Server 2012. Em contrapartida, foi apresentada uma menor vazão pelo mesmo, deixando visível que ele foi mais lento e consumiu mais recursos que o MongoDB. No quesito taxa de erro, ambos executaram todas as operações com sucesso.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	245,10	15,10	0,00	74,00	988,00
MongoDB	307,50	0,60	0,00	48,50	912,00

Tabela 7: Teste de carga da operação *INSERT* executada 1.000 vezes.

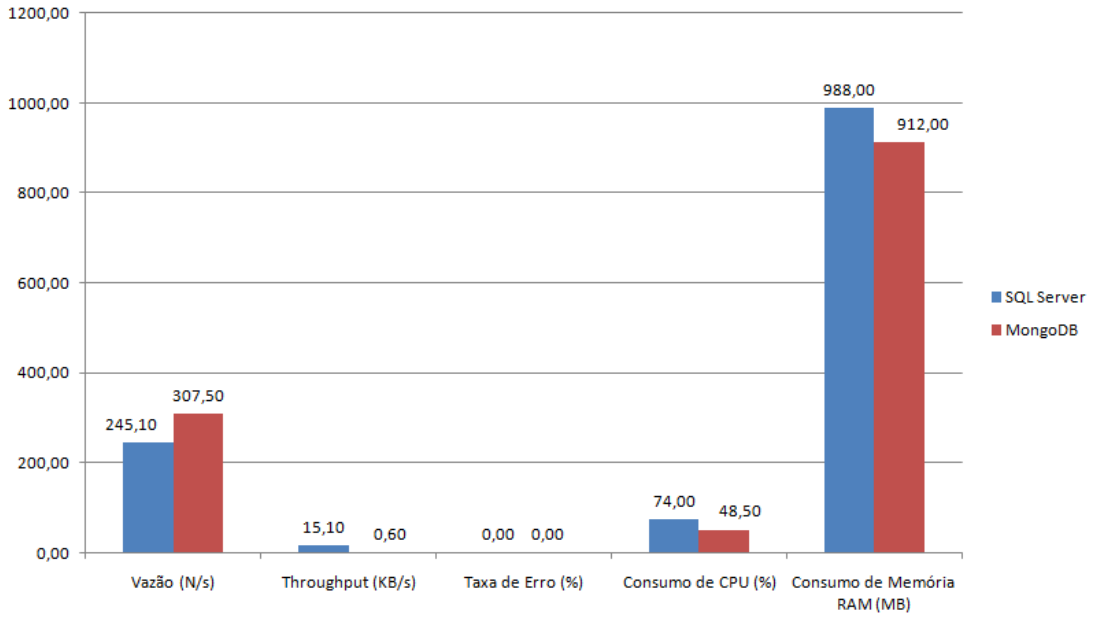


Figura 26: Teste de carga da operação *INSERT* executada 1.000 vezes.

Nos resultados dos testes da operação de *INSERT* executada 10.000 vezes por 1 usuário (Tabela 8 e Figura 27, abaixo) é possível notar um maior *throughput,* vazão e consumo de memória por parte do SQL Server 2012. Em contrapartida, foi apresentado um menor consumo de processamento pelo mesmo, deixando visível que ele foi mais rápido e consumiu mais memória RAM que o MongoDB, porém, ele precisou de menos processamento para tal.

No quesito taxa de erro, ambos executaram todas as operações com sucesso.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	272,90	16,80	0,00	58,50	1059,00
MongoDB	192,80	0,40	0,00	69,50	939,50

Tabela 8: Teste de carga da operação *INSERT* executada 10.000 vezes.

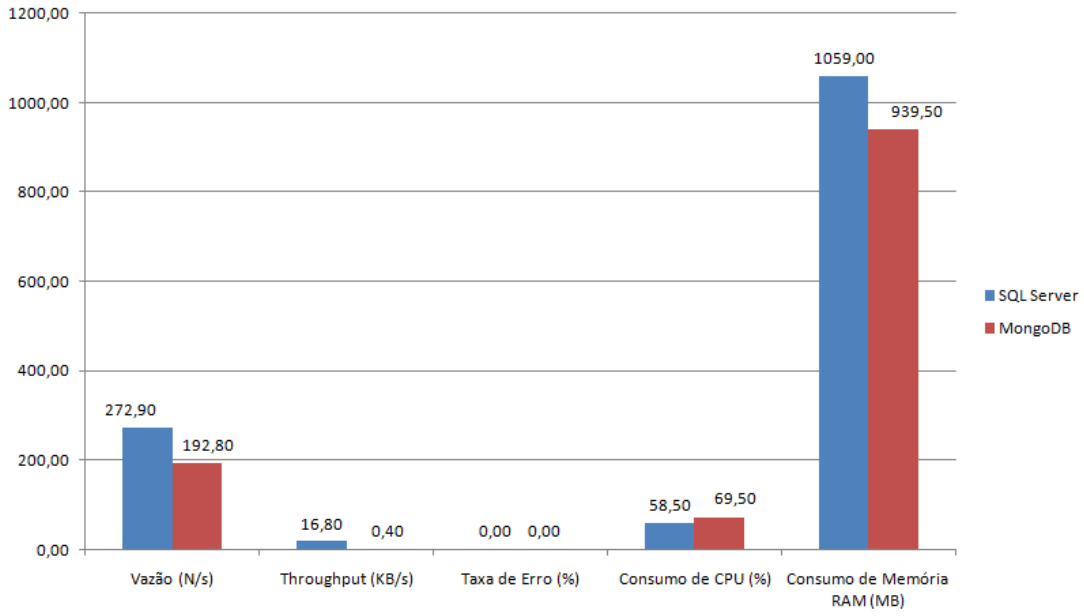


Figura 27: Teste de carga da operação *INSERT* executada 10.000 vezes.

Nos resultados dos testes da operação de *INSERT* executada 100.000 vezes por 1 usuário (Tabela 9 e Figura 28, abaixo) é possível notar um maior *throughput* e consumo de memória por parte do SQL Server 2012. Em contrapartida, foi apresentado um menor consumo de processamento pelo mesmo, deixando visível que ele consumiu mais memória RAM que o MongoDB e necessitou de menos processamento de CPU.

No quesito vazão, a diferença entre os SGBDs foi mínima, deixando visível que ambos tiveram praticamente a mesma velocidade e, no quesito taxa de erro ambos executaram todas as operações com sucesso.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	152,70	9,40	0,00	52,50	1190,00
MongoDB	153,30	0,30	0,00	66,00	996,00

Tabela 9: Teste de carga da operação *INSERT* executada 100.000 vezes.

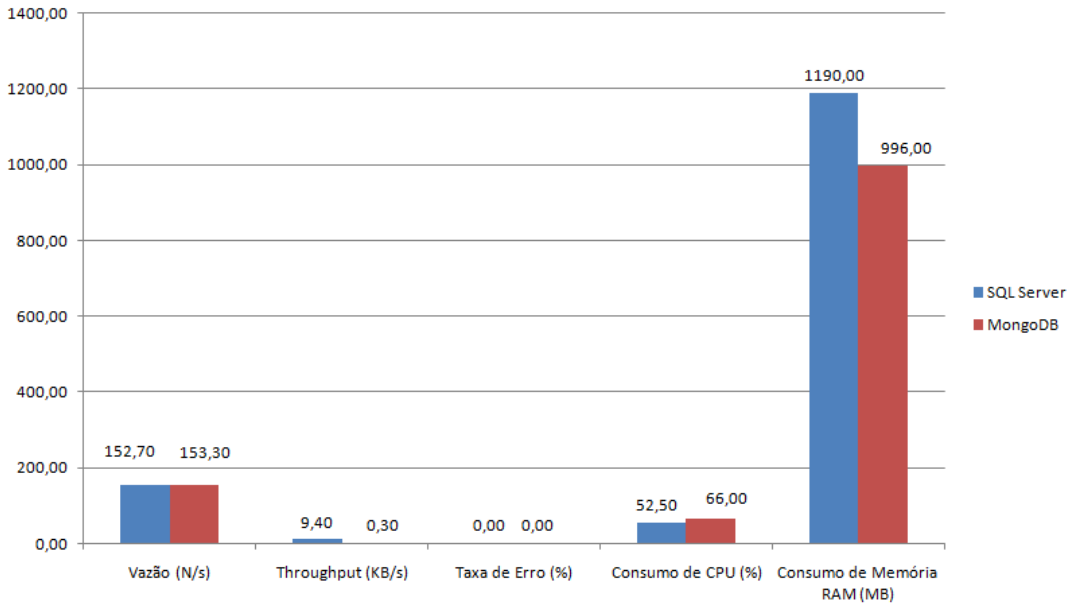


Figura 28: Teste de carga da operação *INSERT* executada 100.000 vezes.

Nos resultados dos testes da operação de *UPDATE* executada 1.000 vezes por 1 usuário (Tabela 10 e Figura 29, abaixo) é possível notar uma menor vazão, um maior consumo de memória e um menor consumo de processamento por parte do SQL Server 2012. Deixando visível que apesar de ter sido mais lento ele consumiu mais memória que o MongoDB.

No quesito consumo de processamento, apesar do MongoDB ter consumido um pouco a mais, ambos utilizaram-se aproximadamente da metade do processamento total do computador. Neste teste também foi possível identificar uma taxa de erro mínima por parte do MongoDB.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	24,60	1,50	0,00	49,50	977,50
MongoDB	175,90	0,40	0,10	55,50	936,00

Tabela 10: Teste de carga da operação *UPDATE* executada 1.000 vezes.

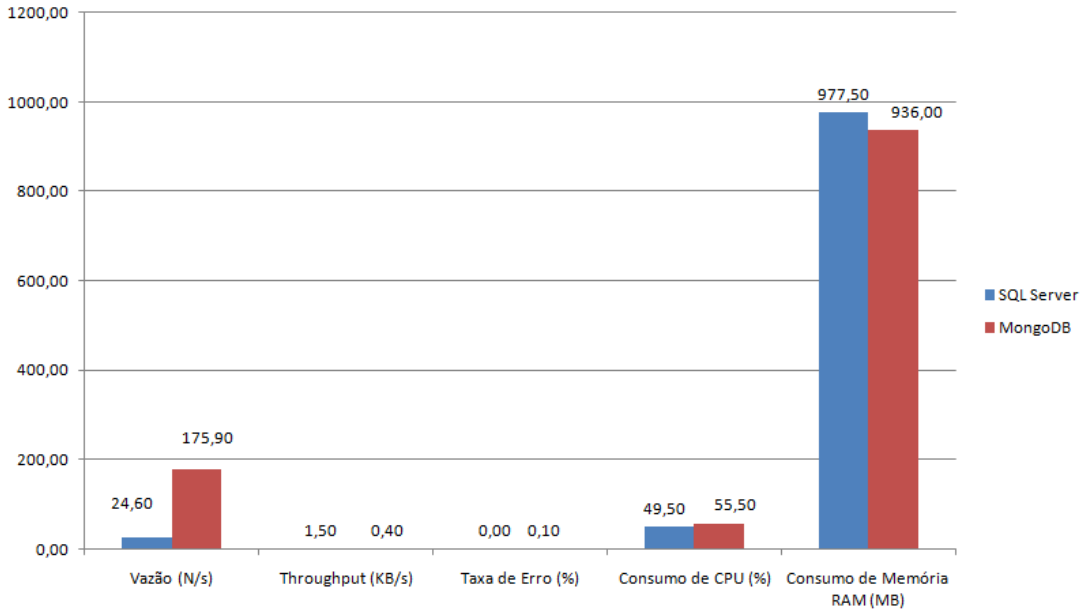


Figura 29: Teste de carga da operação *UPDATE* executada 1.000 vezes.

Nos resultados dos testes da operação de *UPDATE* executada 10.000 vezes por 1 usuário (Tabela 11 e Figura 30, abaixo) é possível notar uma menor vazão, um maior consumo de memória e um menor consumo de processamento por parte do SQL Server 2012. Deixando visível que apesar de ter sido mais lento, ele consumiu mais memória e menos processamento que o MongoDB.

Neste teste, assim como no anterior, também foi possível identificar uma taxa de erro mínima por parte do MongoDB.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	21,80	1,30	0,00	57,00	985,50
MongoDB	111,70	0,20	0,01	69,00	941,00

Tabela 11: Teste de carga da operação *UPDATE* executada 10.000 vezes.

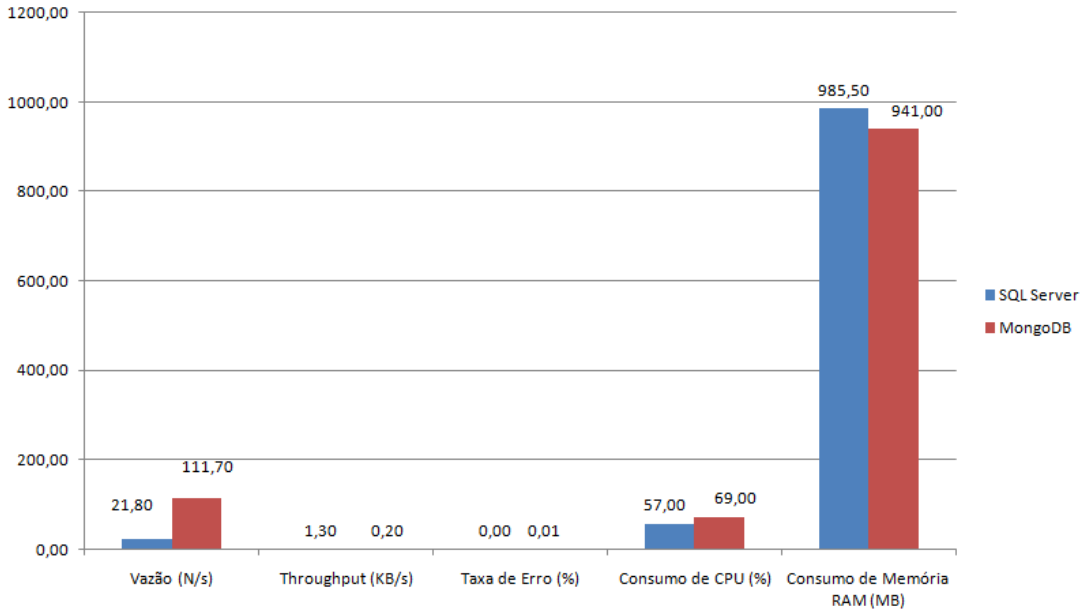


Figura 30: Teste de carga da operação *UPDATE* executada 10.000 vezes.

Nos resultados dos testes da operação de *UPDATE* executada 100.000 vezes por 1 usuário (Tabela 12 e Figura 31, abaixo) é possível notar uma menor vazão, maior consumo de memória e um menor consumo de processamento por parte do SQL Server 2012. Deixando visível que além de ter sido mais lento, ele consumiu mais memória e menos processamento que o MongoDB.

Neste teste foi possível identificar que o MongoDB teve uma taxa de erro de 0% e, se isso for comparado com os testes de 1.000 e 10.000 execuções, pode ser notado que quanto maior a quantidade de dados ao qual ele foi submetido, menor foi a sua taxa de erro.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	18,50	1,10	0,00	64,50	1170,00
MongoDB	126,00	0,20	0,00	68,00	934,50

Tabela 12: Teste de carga da operação *UPDATE* executada 100.000 vezes.

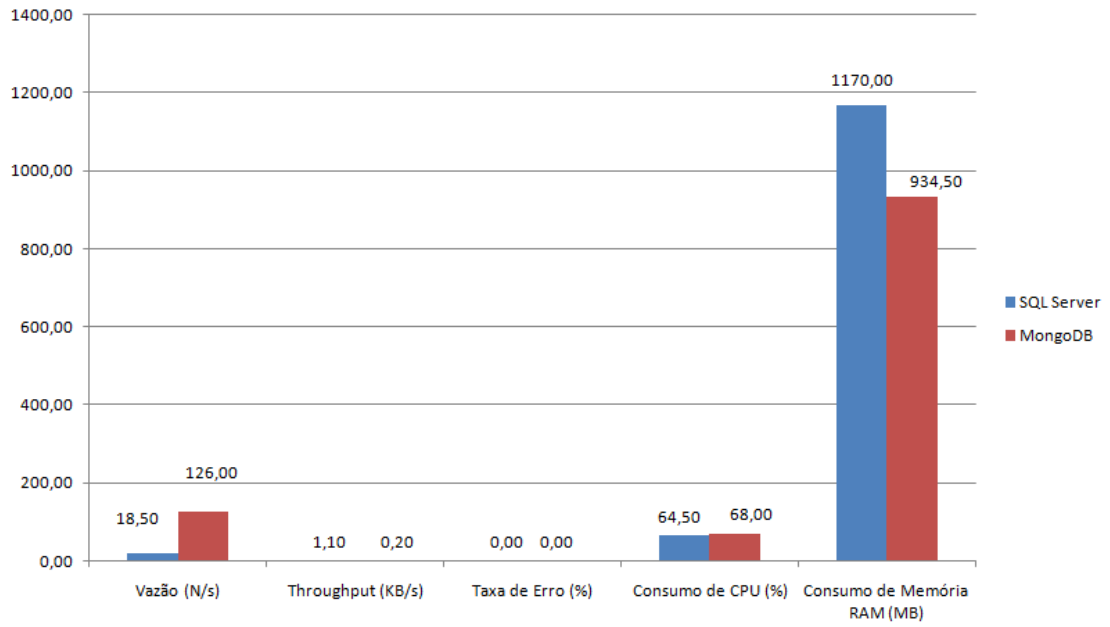


Figura 31: Teste de carga da operação *UPDATE* executada 100.000 vezes.

Nos resultados dos testes da operação de *DELETE* executada 1.000 vezes por 1 usuário (Tabela 13 e Figura 32, abaixo) é possível notar uma menor vazão e, um maior consumo de memória e processamento por parte do SQL Server 2012. Deixando visível que apesar de ter sido mais lento, ele consumiu mais memória e mais processamento que o MongoDB.

No quesito taxa de erro, ambos executaram todas as operações com sucesso.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	8,80	0,50	0,00	54,50	1000,50
MongoDB	194,50	0,40	0,00	34,50	917,00

Tabela 13: Teste de carga da operação *DELETE* executada 1.000 vezes.

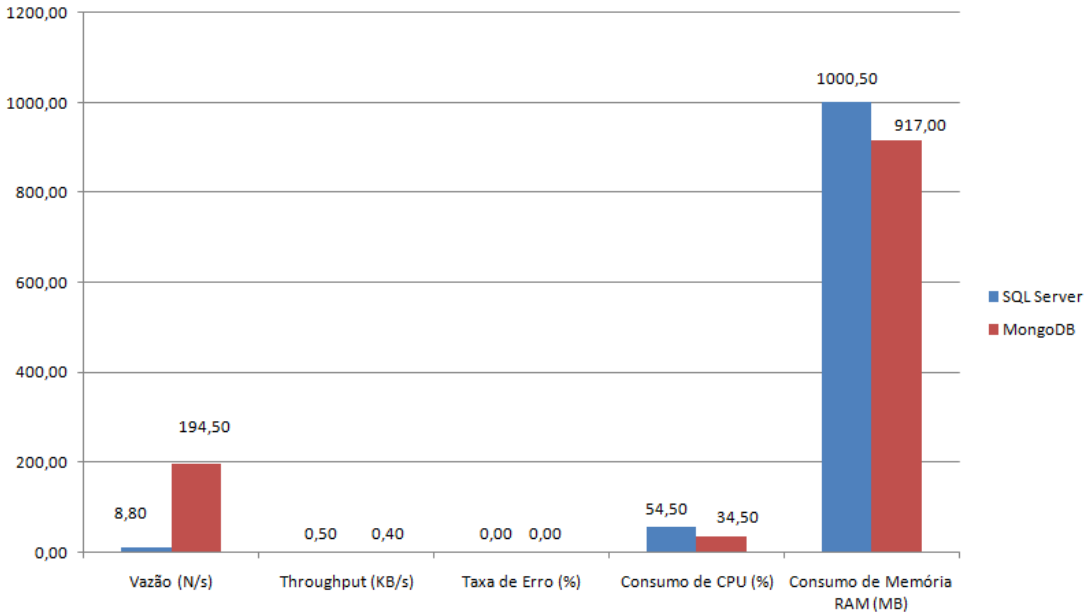


Figura 32: Teste de carga da operação *DELETE* executada 1.000 vezes.

Nos resultados dos testes da operação de *DELETE* executada 10.000 vezes por 1 usuário (Tabela 14 e Figura 33, abaixo) é possível notar um menor consumo de memória e uma diferença mínima - para menos - de consumo de processamento por parte do SQL Server 2012 em relação ao MongoDB. Porém, a sua taxa de vazão também foi muito inferior à do MongoDB, indicando que ele foi muito mais lento na execução das operações.

No quesito taxa de erro, ambos executaram todas as operações com sucesso.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	7,80	0,50	0,00	65,50	841,00
MongoDB	199,00	0,40	0,00	66,00	930,00

Tabela 14: Teste de carga da operação *DELETE* executada 10.000 vezes.

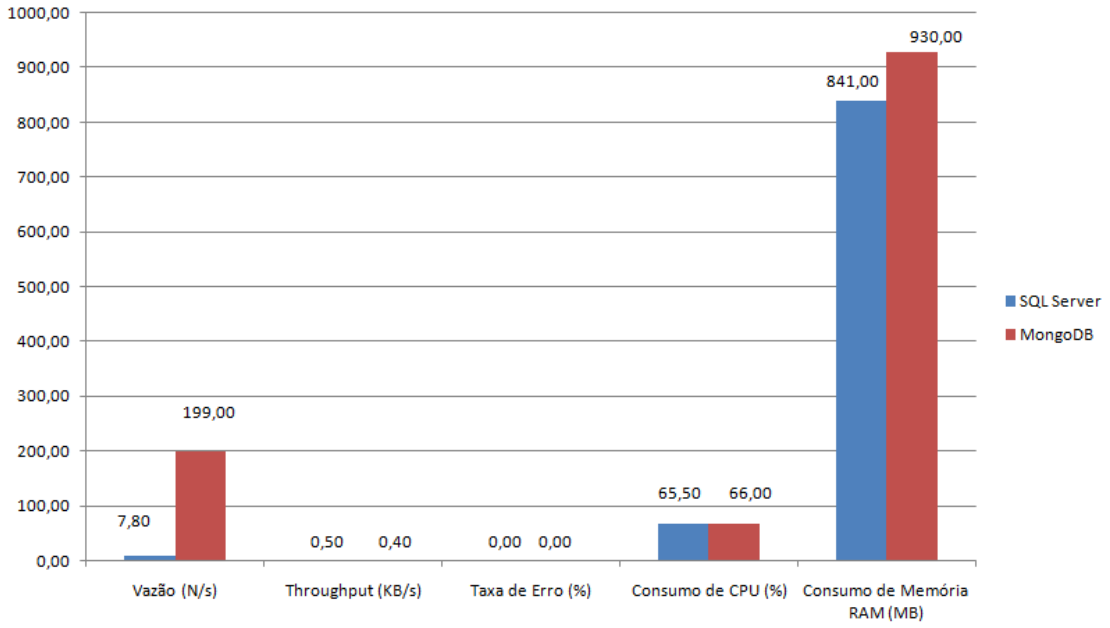


Figura 33: Teste de carga da operação *DELETE* executada 10.000 vezes.

Nos resultados dos testes da operação de *DELETE* executada 100.000 vezes por 1 usuário (Tabela 15 e Figura 34, abaixo) é possível notar um menor consumo de memória e um menor consumo de processamento por parte do SQL Server 2012 em relação ao MongoDB. Porém, a sua taxa de vazão também foi inferior à do MongoDB, indicando que ele foi mais lento na execução das operações.

No quesito taxa de erro, o MongoDB apresentou uma taxa mínima de erro e o SQL Server 2012 executou todas as operações corretamente.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	8,10	0,50	0,00	60,50	828,00
MongoDB	95,20	0,20	0,01	70,50	907,50

Tabela 15: Teste de carga da operação *DELETE* executada 100.000 vezes.

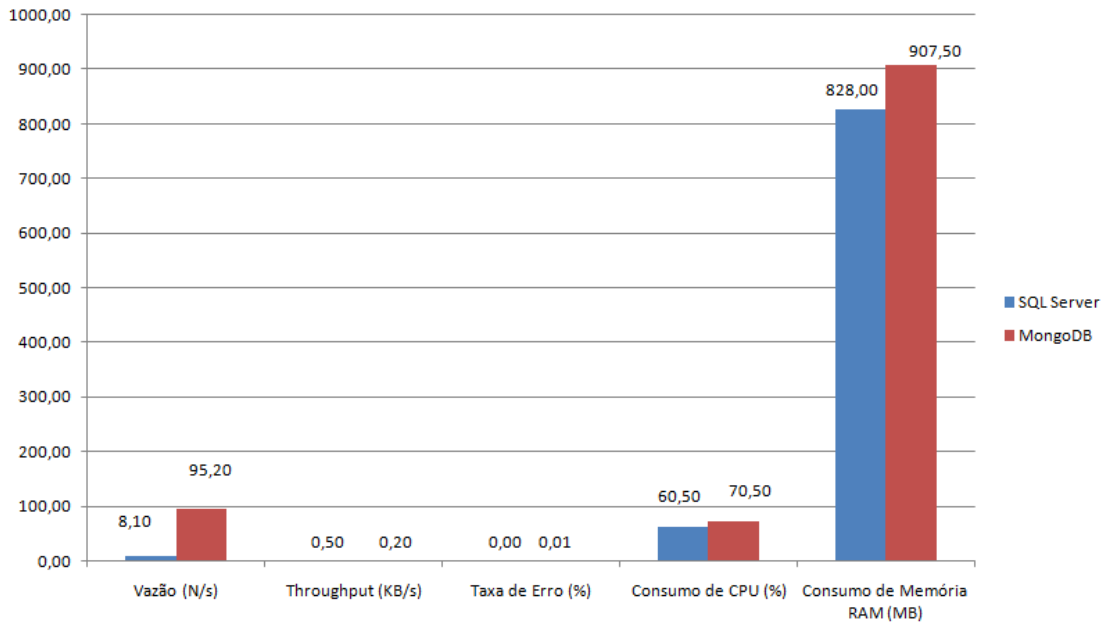


Figura 34: Teste de carga da operação *DELETE* executada 100.000 vezes.

Nos resultados dos testes da operação de *SELECT* executada 1.000 vezes por 1 usuário (Tabela 16 e Figura 35, abaixo) é possível notar um maior consumo de memória e um maior consumo de processamento por parte do SQL Server 2012 em relação ao MongoDB. Porém, a sua taxa de vazão foi inferior à do MongoDB, indicando que ele foi mais lento na execução das operações e consumiu mais recursos para isso.

No quesito taxa de erro, o MongoDB apresentou uma taxa mínima de erro e o SQL Server 2012 executou todas as operações corretamente.

Neste teste também se deve ressaltar a diferença assombrosa de *throughput* entre ambos, caso que não havia acontecido com tanta diferença nos testes anteriores. Isso provavelmente se deve à diferença entre ambos os SGBDs quanto as suas formas de armazenamento de dados.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	58,20	5751,80	0,00	62,00	990,50
MongoDB	201,10	112,00	0,10	52,00	932,00

Tabela 16: Teste de carga da operação *SELECT* executada 1.000 vezes.

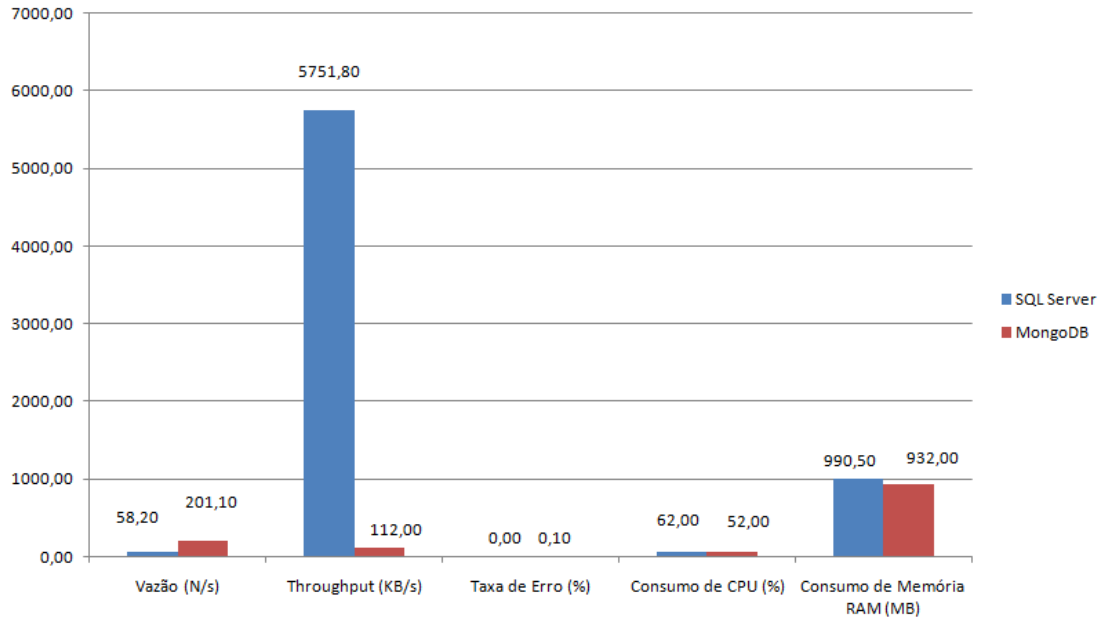


Figura 35: Teste de carga da operação *SELECT* executada 1.000 vezes.

Nos resultados dos testes da operação de *SELECT* executada 10.000 vezes por 1 usuário (Tabela 17 e Figura 36, abaixo) é possível notar um maior consumo de memória por parte do SQL Server 2012 em relação ao MongoDB e um consumo de processamento praticamente igual entre ambos. Porém, a taxa de vazão do SQL Server 2012 foi muito inferior à do MongoDB, indicando que ele foi muito mais lento na execução das operações e consumiu mais memória para isso.

No quesito taxa de erro, o MongoDB apresentou uma taxa mínima de erro e o SQL Server 2012 executou todas as operações corretamente.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	3,90	4031,40	0,00	57,00	1135,00
MongoDB	207,00	115,40	0,01	57,50	957,00

Tabela 17: Teste de carga da operação *SELECT* executada 10.000 vezes.

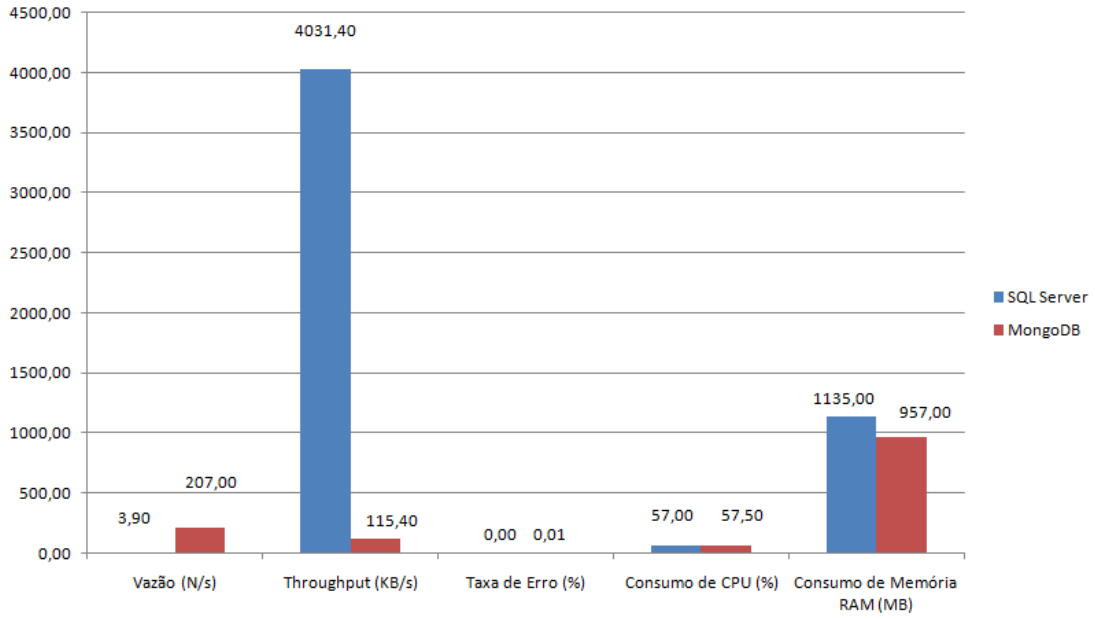


Figura 36: Teste de carga da operação *SELECT* executada 10.000 vezes.

Nos resultados dos testes da operação de *SELECT* executada 100.000 vezes por 1 usuário (Tabela 18 e Figura 37, abaixo) é possível notar um maior consumo de memória e um menor consumo de processamento por parte do SQL Server 2012 em relação ao MongoDB. Porém, a taxa de vazão do SQL Server 2012 foi muito inferior à do MongoDB, indicando que ele foi muito mais lento na execução das operações e consumiu mais memória para isso.

No quesito taxa de erro, ambos executaram todas as operações com sucesso.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	0,12	1353,40	0,00	63,50	1295,00
MongoDB	150,50	83,90	0,00	71,00	826,00

Tabela 18: Teste de carga da operação *SELECT* executada 100.000 vezes.

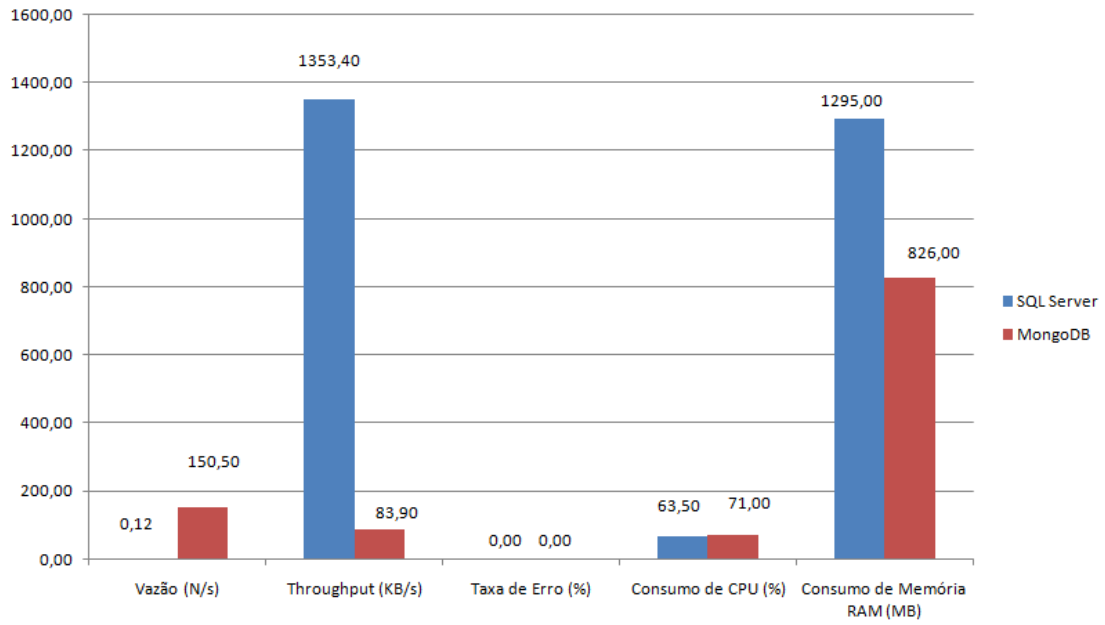


Figura 37: Teste de carga da operação *SELECT* executada 100.000 vezes.

4.3.2 Testes de *Stress*

Nos resultados dos testes da operação de *INSERT* executada 1 vez por 1.000 usuários (Tabela 19 e Figura 38, abaixo) é possível notar uma maior vazão e um menor consumo de memória e processamento por parte do SQL Server 2012, deixando visível que ele foi mais rápido e consumiu menos recursos que o MongoDB.

No quesito taxa de erro, enquanto o SQL Server 2012 executou todas as operações com sucesso, o MongoDB executou somente 21,20% das operações com sucesso. O maior motivo da alta taxa de erro por parte do MongoDB se deve ao fato

de ter sido atingido o limite máximo de conexões ao SGBD, com isso, todas as conexões posteriores foram simplesmente descartadas.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	780,60	48,00	0,00	67,00	1100,00
MongoDB	596,70	31,80	78,80	91,00	1220,00

Tabela 19: Teste de *stress* da operação *INSERT* executada 1.000 vezes.

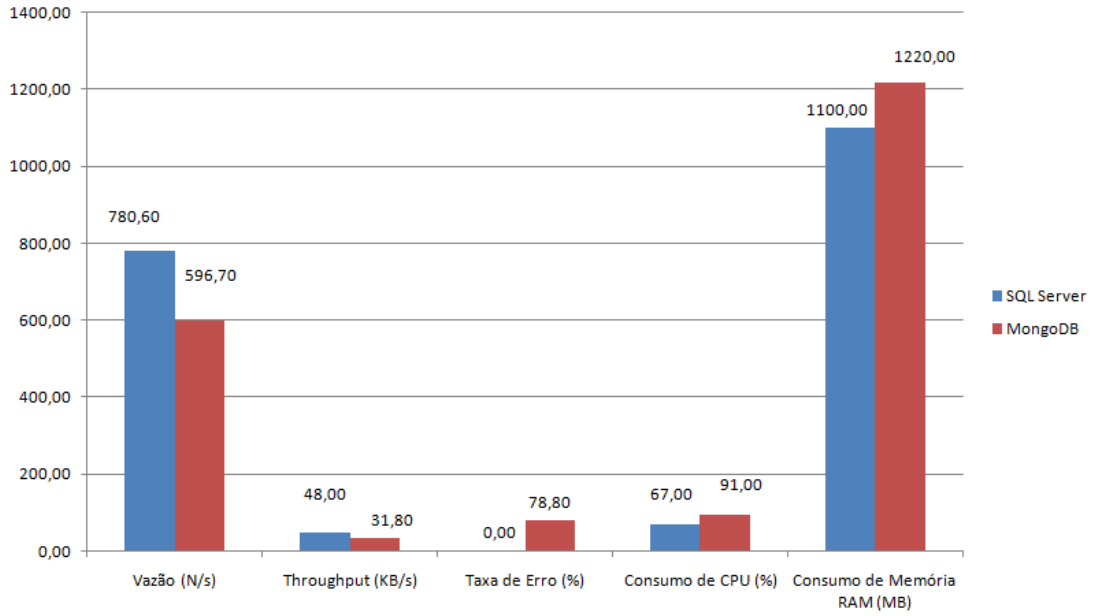


Figura 38: Teste de *stress* da operação *INSERT* executada 1.000 vezes.

Nos resultados dos testes da operação de *INSERT* executada 10 vezes por 1.000 usuários (Tabela 20 e Figura 39, abaixo) é possível notar uma menor vazão e um menor consumo de memória e processamento por parte do SQL Server 2012, deixando visível que ele foi mais lento e consumiu menos recursos que o MongoDB.

No quesito taxa de erro, enquanto o SQL Server 2012 executou 99,83% das operações com sucesso, o MongoDB executou somente 26,46% das operações.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	766,20	47,30	0,17	89,50	974,50
MongoDB	941,10	47,10	73,54	97,00	1215,00

Tabela 20: Teste de *stress* da operação *INSERT* executada 10.000 vezes.

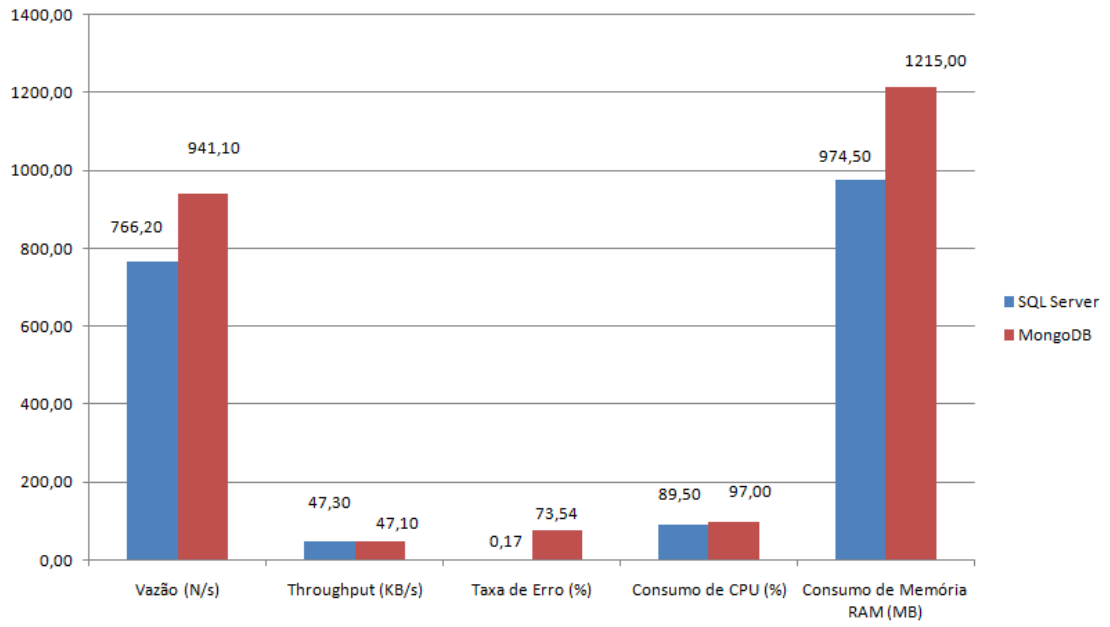


Figura 39: Teste de *stress* da operação *INSERT* executada 10.000 vezes.

Nos resultados dos testes da operação de *INSERT* executada 100 vezes por 1.000 usuários (Tabela 21 e Figura 40, abaixo) é possível notar uma maior vazão, um maior consumo de memória e um processamento praticamente igual ao do MongoDB por parte do SQL Server 2012. Deixando visível que ele teve mais que o dobro de velocidade que o MongoDB e consumiu mais memória.

No quesito taxa de erro, enquanto o SQL Server 2012 executou 99,80% das operações com sucesso, o MongoDB executou somente 41,90% das operações.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	748,70	46,40	0,20	73,00	1095,00
MongoDB	347,00	13,90	58,10	74,50	977,50

Tabela 21: Teste de *stress* da operação *INSERT* executada 100.000 vezes.

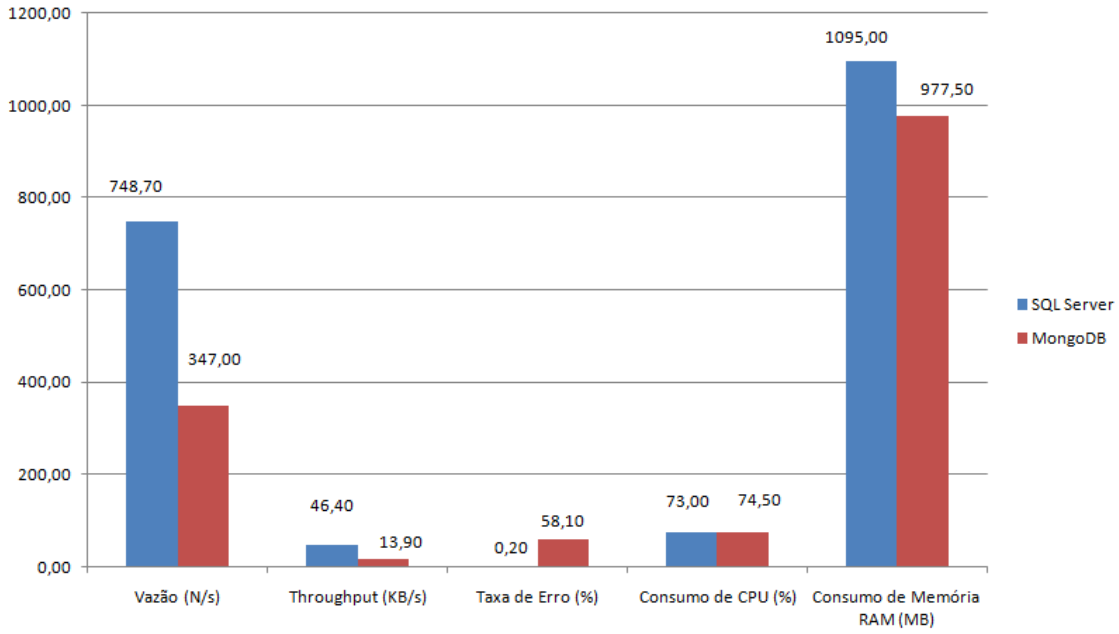


Figura 40: Teste de *stress* da operação *INSERT* executada 100.000 vezes.

Nos resultados dos testes da operação de *UPDATE* executada 1 vez por 1.000 usuários (Tabela 22 e Figura 41, abaixo) é possível notar uma menor vazão, um menor consumo de memória e um consumo de processamento um pouco mais alto por parte do SQL Server 2012 em relação ao MongoDB. Deixando visível que ele foi 5,48 vezes mais lento que o MongoDB e consumiu menos memória.

Porém, no quesito taxa de erro, enquanto o SQL Server 2012 executou 49,80% das operações com sucesso, o MongoDB executou somente 24,40% das operações. O motivo da taxa de erro do SQL Server ter crescido tanto neste teste e nos testes seguintes é que, várias das transações concorrentes foram descartadas pelo SGBD enquanto existia alguma já sendo executada.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	88,30	5,80	50,20	97,50	1095,00
MongoDB	483,80	24,80	75,60	94,00	1220,00

Tabela 22: Teste de *stress* da operação *UPDATE* executada 1.000 vezes.

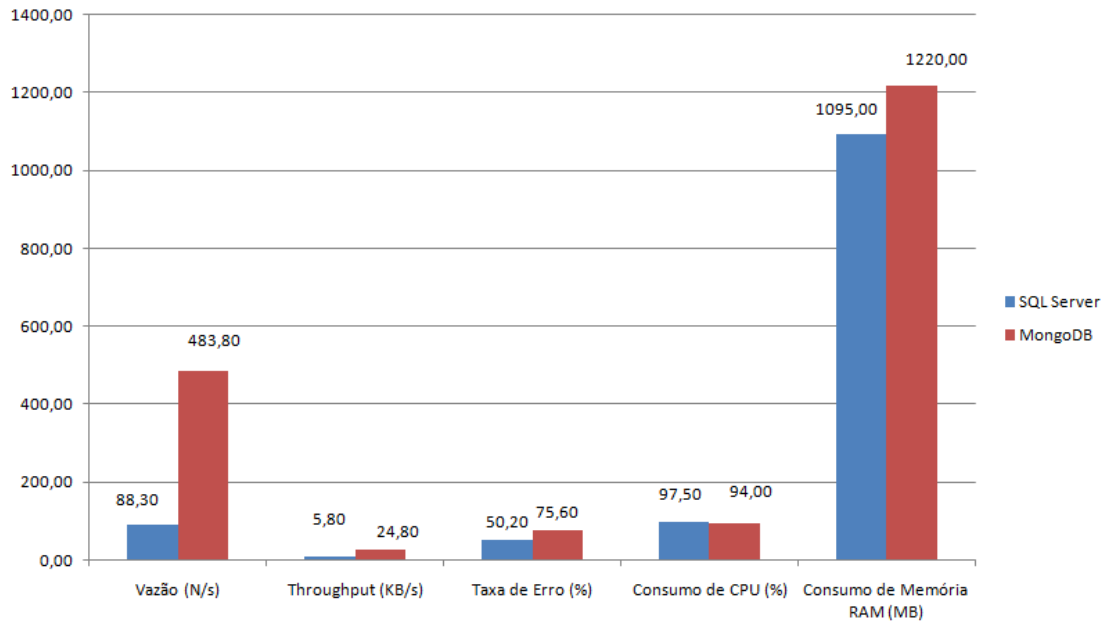


Figura 41: Teste de *stress* da operação *UPDATE* executada 1.000 vezes.

Nos resultados dos testes da operação de *UPDATE* executada 10 vezes por 1.000 usuários (Tabela 23 e Figura 42, abaixo) é possível notar uma menor vazão, um menor consumo de memória e um consumo de processamento um pouco mais alto por parte do SQL Server 2012 em relação ao MongoDB. Deixando visível que ele foi 9,78 vezes mais lento que o MongoDB e consumiu menos memória.

Porém, no quesito taxa de erro, enquanto o SQL Server 2012 executou 46,14% das operações com sucesso, o MongoDB executou somente 16,19% das operações.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	97,10	6,40	53,86	99,00	1010,00
MongoDB	949,60	53,90	83,81	97,00	1200,00

Tabela 23: Teste de *stress* da operação *UPDATE* executada 10.000 vezes.

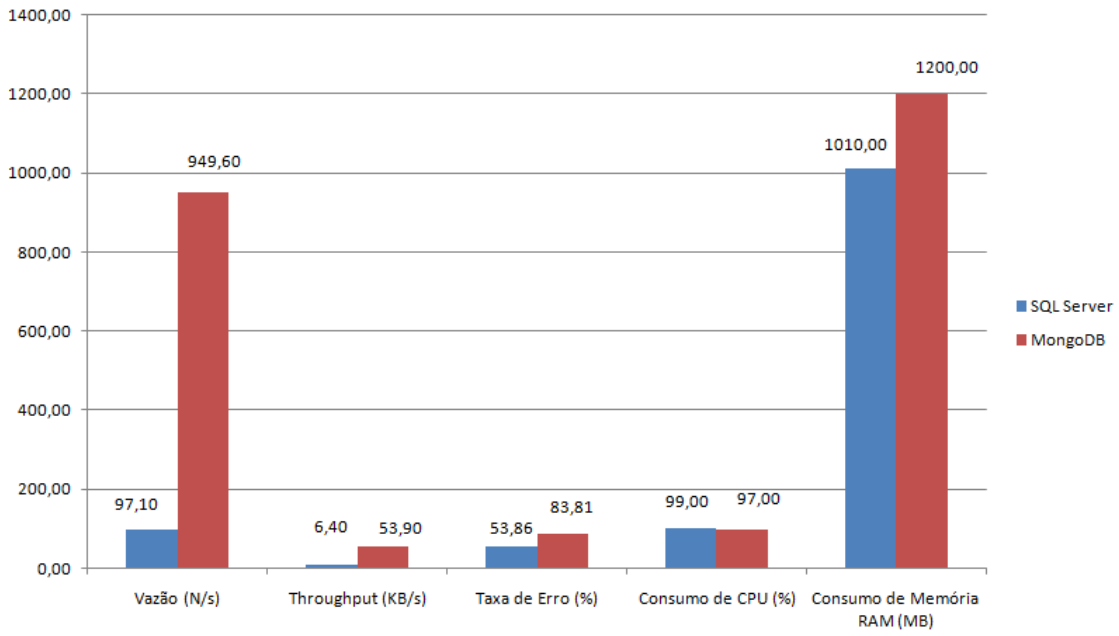


Figura 42: Teste de *stress* da operação *UPDATE* executada 10.000 vezes.

Nos resultados dos testes da operação de *UPDATE* executada 100 vezes por 1.000 usuários (Tabela 24 e Figura 43, abaixo) é possível notar uma menor vazão, um menor consumo de processamento e um maior consumo de memória por parte do SQL Server 2012 em relação ao MongoDB. Deixando visível que ele foi 3,17 vezes mais lento que o MongoDB e consumiu mais memória.

Desta vez, no quesito taxa de erro ambos ficaram acima dos 70%, com o SQL Server 2012 executando apenas 29,44% das operações com sucesso e o MongoDB executando somente 19,91% das operações.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	88,50	5,80	70,56	74,00	1305,00
MongoDB	279,70	15,20	80,09	92,00	1195,00

Tabela 24: Teste de *stress* da operação *UPDATE* executada 100.000 vezes.

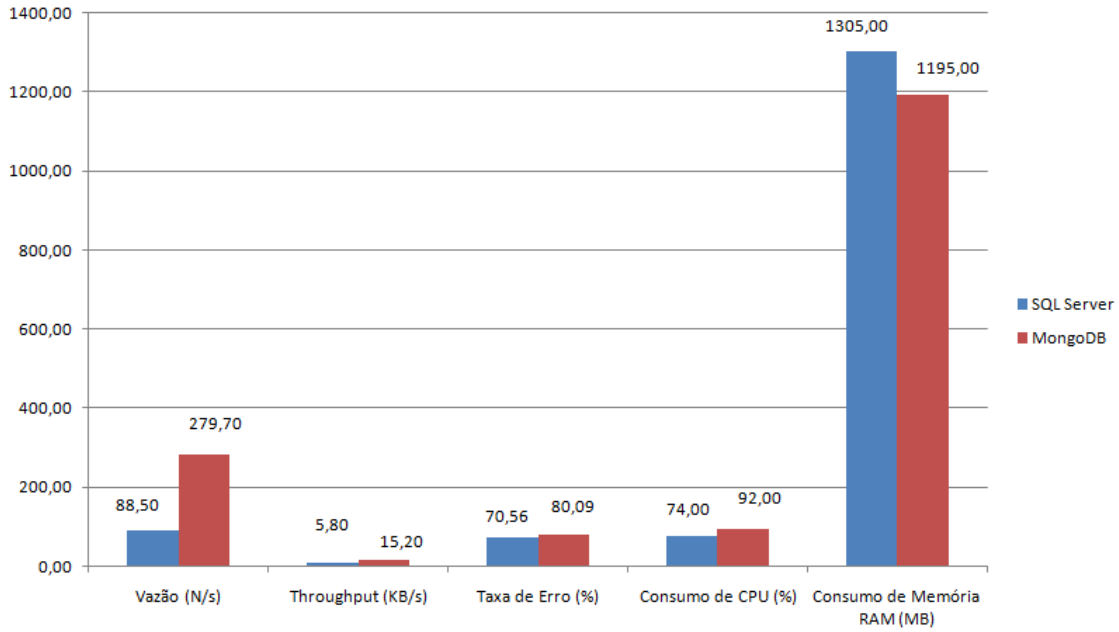


Figura 43: Teste de *stress* da operação *UPDATE* executada 100.000 vezes.

Nos resultados dos testes da operação de *DELETE* executada 1 vez por 1.000 usuários (Tabela 25 e Figura 44, abaixo) é possível notar uma menor vazão e um menor consumo de processamento e memória por parte do SQL Server 2012 em relação ao MongoDB. Deixando visível que ele foi aproximadamente 6 vezes mais lento que o MongoDB e consumiu menos recursos.

Um fato inédito no quesito taxa de erro dos testes de stress é que, na operação de *DELETE* o SQL Server 2012 executou apenas 0,4% das operações com sucesso enquanto que o MongoDB executou 23,3% das operações. Sendo esta, a primeira vez que temos um resultado onde o MongoDB tem uma taxa de erro inferior a do SQL Server 2012 nos testes de *stress*.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	82,20	5,80	99,60	29,00	1175,00
MongoDB	495,00	27,20	76,70	90,50	1220,00

Tabela 25: Teste de *stress* da operação *DELETE* executada 1.000 vezes.

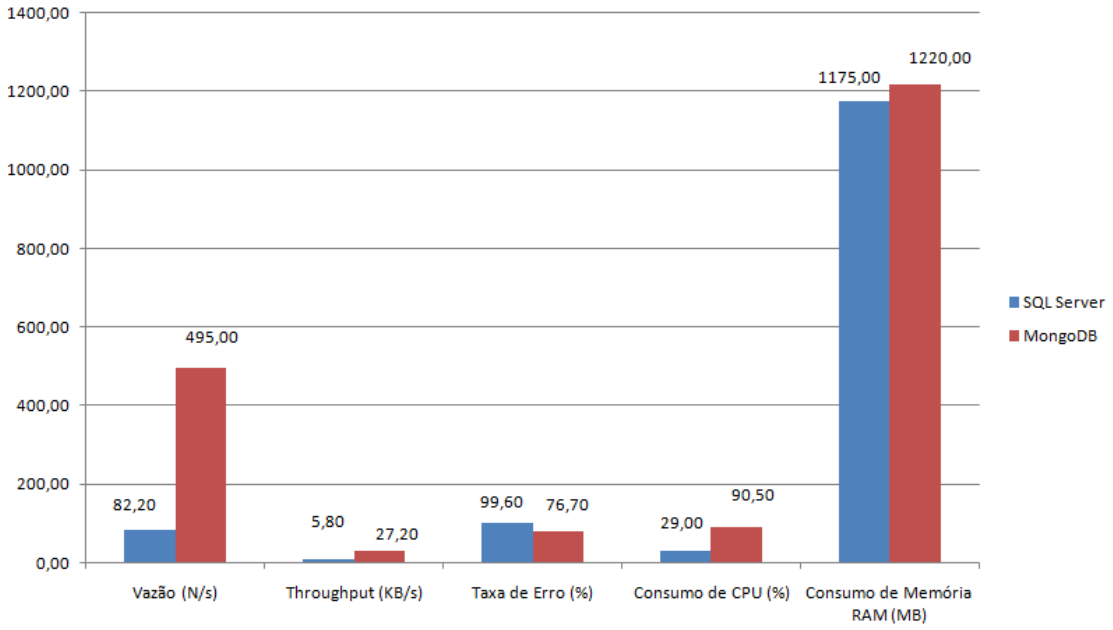


Figura 44: Teste de *stress* da operação *DELETE* executada 1.000 vezes.

Nos resultados dos testes da operação de *DELETE* executada 10 vezes por 1.000 usuários (Tabela 26 e Figura 45, abaixo) é possível notar uma menor vazão e um menor consumo de processamento e memória por parte do SQL Server 2012 em relação ao MongoDB. Deixando visível que ele foi aproximadamente 11,5 vezes mais lento que o MongoDB e consumiu menos recursos.

No quesito taxa de erro, o SQL Server 2012 executou apenas 0,53% das operações com sucesso enquanto que, o MongoDB executou 20,57% das operações.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	89,30	6,80	99,47	59,00	991,00
MongoDB	1026,60	55,30	79,43	97,00	1200,00

Tabela 26: Teste de *stress* da operação *DELETE* executada 10.000 vezes.

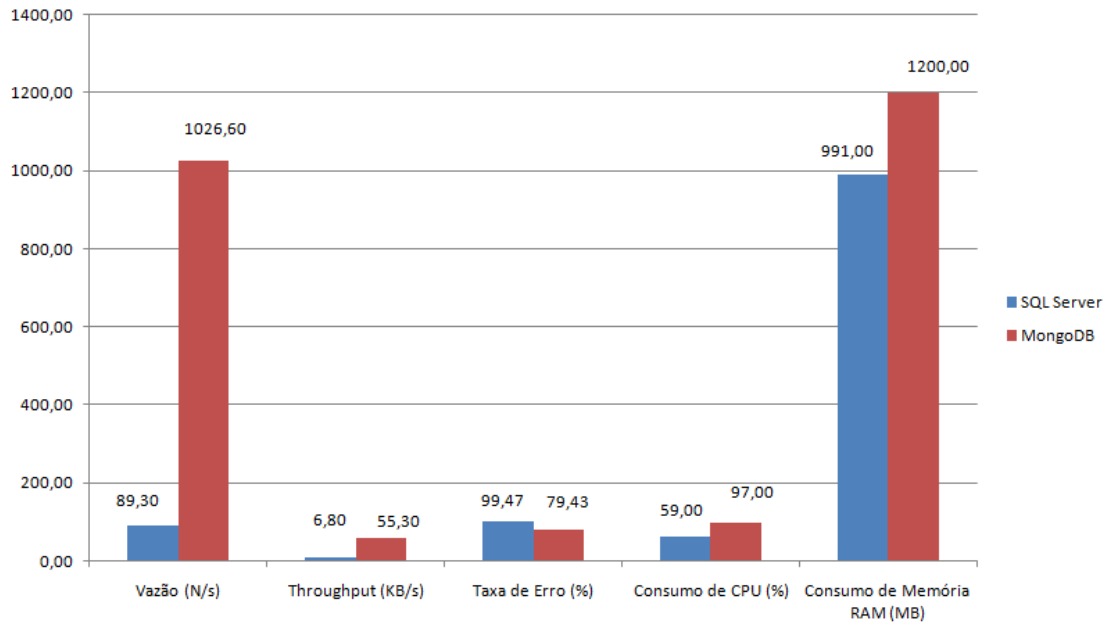


Figura 45: Teste de *stress* da operação *DELETE* executada 10.000 vezes.

Nos resultados dos testes da operação de *DELETE* executada 100 vezes por 1.000 usuários (Tabela 27 e Figura 46, abaixo) é possível notar uma menor vazão e um menor consumo de processamento e memória por parte do SQL Server 2012 em relação ao MongoDB. Deixando visível que ele foi aproximadamente 3,7 vezes mais lento que o MongoDB e consumiu menos recursos.

No quesito taxa de erro, o SQL Server 2012 executou apenas 0,22% das operações com sucesso enquanto que, o MongoDB executou 29,44% das operações.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	79,90	5,20	99,78	50,00	1145,00
MongoDB	295,20	14,20	70,56	93,00	1310,00

Tabela 27: Teste de *stress* da operação *DELETE* executada 100.000 vezes.

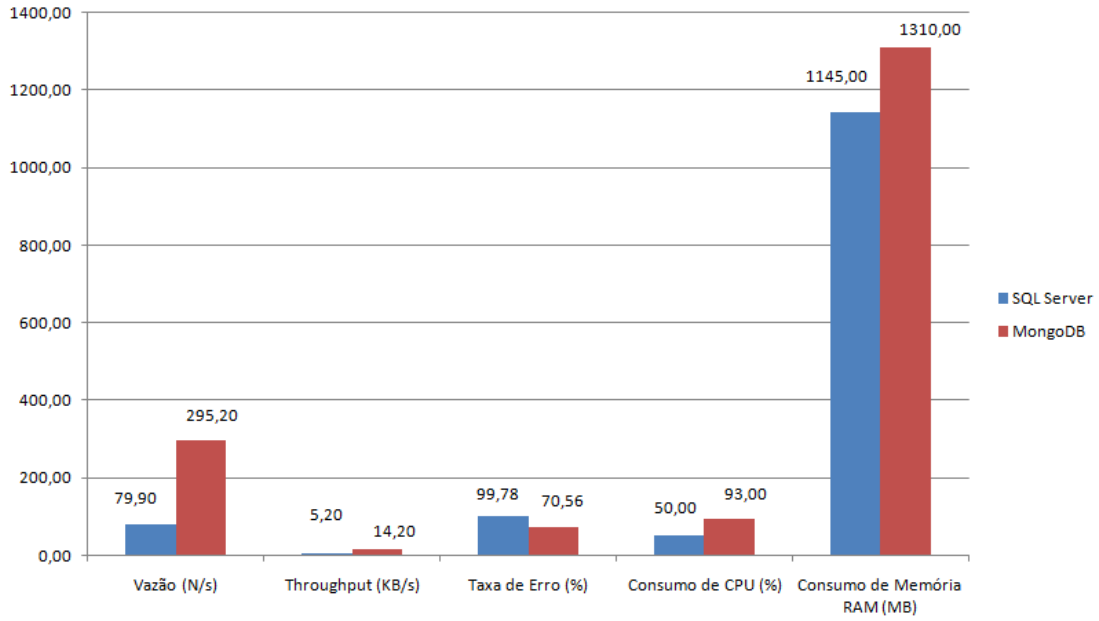


Figura 46: Teste de *stress* da operação *DELETE* executada 100.000 vezes.

Nos resultados dos testes da operação de *SELECT* executada 1 vez por 1.000 usuários (Tabela 28 e Figura 47, abaixo) é possível notar uma menor vazão, um menor consumo de memória e um maior consumo de processamento por parte do SQL Server 2012 em relação ao MongoDB. Deixando visível que ele foi aproximadamente 5,7 vezes mais lento que o MongoDB e consumiu menos memória e mais CPU.

No quesito taxa de erro, o SQL Server 2012 executou 93,80% das operações com sucesso enquanto que, o MongoDB executou apenas 37% das operações.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	88,60	7855,90	6,20	100,00	1080,00
MongoDB	498,80	124,10	63,00	81,50	1240,00

Tabela 28: Teste de *stress* da operação *SELECT* executada 1.000 vezes.

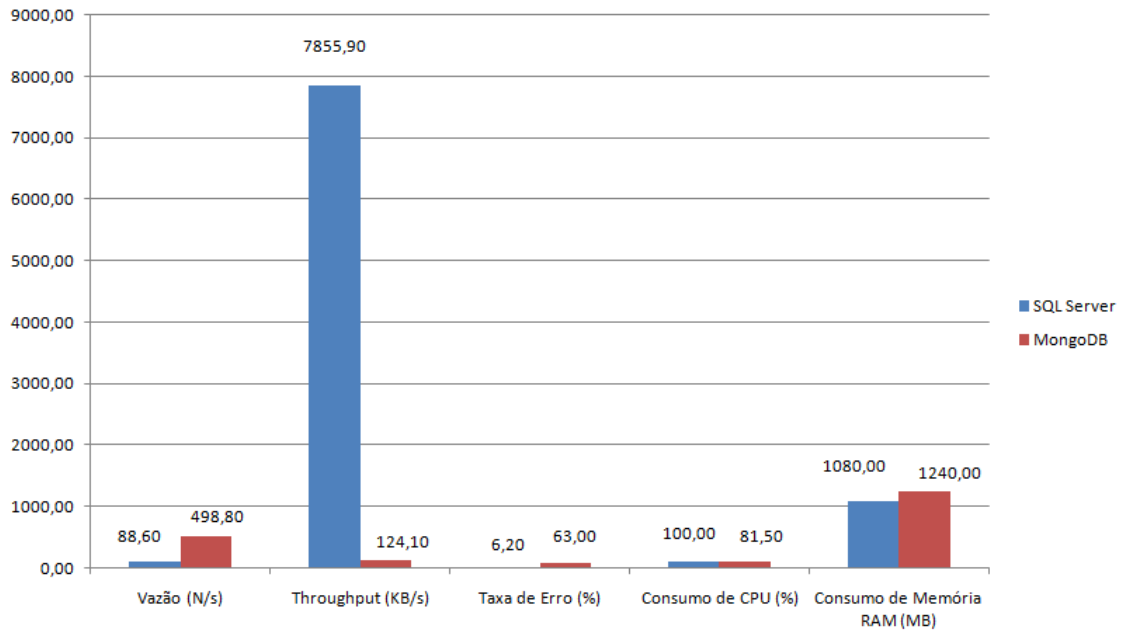


Figura 47: Teste de *stress* da operação *SELECT* executada 1.000 vezes.

Nos resultados dos testes da operação de *SELECT* executada 10 vezes por 1.000 usuários (Tabela 29 e Figura 48, abaixo) é possível notar uma menor vazão, um menor consumo de processamento e um consumo de memória praticamente igual ao do MongoDB por parte do SQL Server 2012. Deixando visível que ele foi aproximadamente 17,15 vezes mais lento que o MongoDB e consumiu menos processamento de CPU.

No quesito taxa de erro, o SQL Server 2012 executou 89,70% das operações com sucesso enquanto que, o MongoDB executou apenas 23,60% das operações.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	61,20	8569,40	10,30	75,00	1161,50
MongoDB	1049,00	192,00	76,40	98,00	1175,00

Tabela 29: Teste de *stress* da operação *SELECT* executada 10.000 vezes.

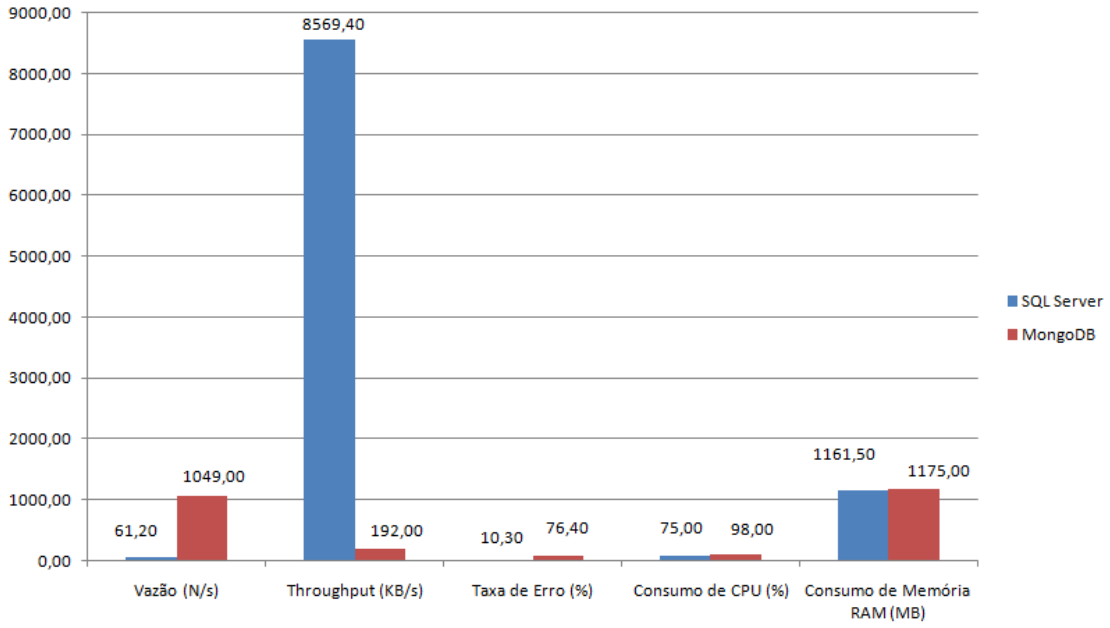


Figura 48: Teste de *stress* da operação *SELECT* executada 10.000 vezes.

Nos resultados dos testes da operação de *SELECT* executada 100 vezes por 1.000 usuários (Tabela 30 e Figura 49, abaixo) é possível notar uma menor vazão, um menor consumo de processamento e um maior consumo de memória por parte do SQL Server 2012. Deixando visível que ele foi aproximadamente 465,30 vezes mais lento que o MongoDB e consumiu mais memória.

No quesito taxa de erro, o SQL Server 2012 executou apenas 0,06% das operações com sucesso enquanto que o MongoDB executou 55,50% das operações.

SGBD	Vazão (N/s)	Throughput (KB/s)	Taxa de Erro (%)	Consumo de CPU (%)	Consumo de Memória RAM (MB)
SQL Server	0,74	4,20	99,94	69,00	1525,00
MongoDB	342,00	116,10	44,50	74,50	1310,00

Tabela 30: Teste de *stress* da operação *SELECT* executada 100.000 vezes.

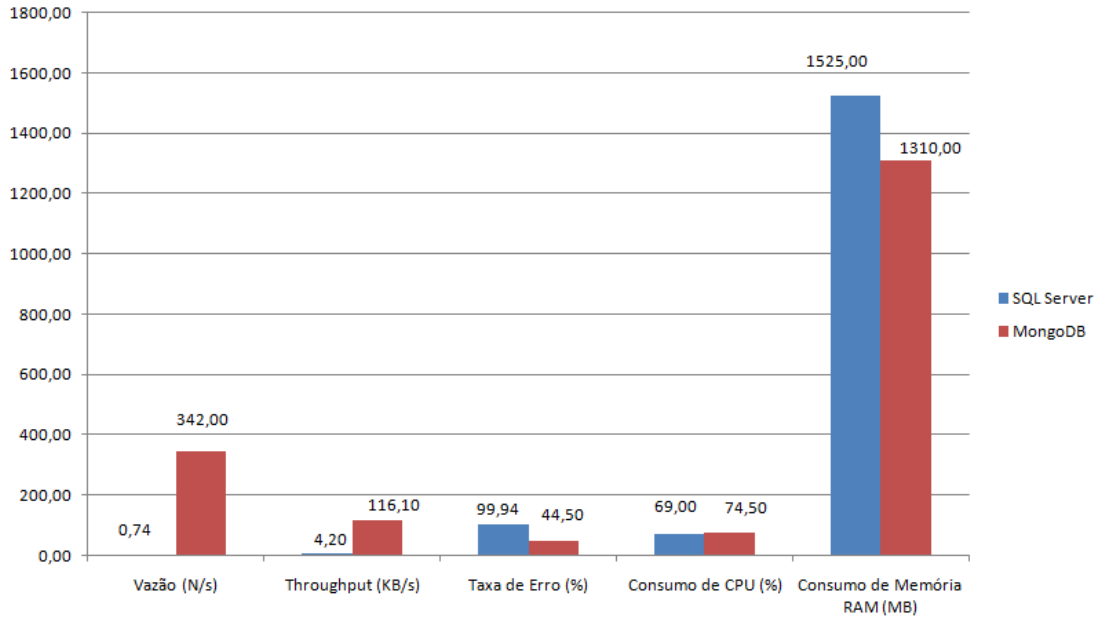


Figura 49: Teste de *stress* da operação *SELECT* executada 100.000 vezes.

5 CONCLUSÃO E PERSPECTIVAS FUTURAS

5.1 CONCLUSÃO

Como pôde ser visto no capítulo anterior, nos testes de carga o SQL Server 2012 consumiu mais memória RAM em relação ao MongoDB em quase todas as operações e, em contrapartida, teve um consumo de processamento de CPU mais baixo.

No quesito Vazão, o MongoDB demonstrou ter uma superioridade ao SQL Server 2012 em praticamente todas as operações, com isso, pode-se inferir que ele foi mais rápido que o SGBD relacional.

No quesito *throughput*, o SQL Server 2012 demonstrou uma taxa superior em todas as operações em comparação à tecnologia NoSQL. Como pôde ser visto em capítulos anteriores deste trabalho, isso pode ser explicado pela estrutura dos dados de cada tecnologia. Uma vez que, os dados no formato JSON do MongoDB ocupam muito menos espaço que as tabelas do SGBDR SQL Server 2012.

No quesito Taxa de Erro, pode-se observar que o SQL Server 2012 concluiu todas as operações com sucesso, enquanto que, o MongoDB apresentou uma taxa de erro quase que insignificante. Mas, também é notável que na maioria dos casos, essa taxa de erro tende de diminuir conforme é aumentada a quantidade de operações ao qual ele é submetido.

Nos testes de stress o SQL Server 2012 consumiu menos memória RAM e menos processamento de CPU em relação ao MongoDB em quase todas as operações.

No quesito Vazão, o MongoDB demonstrou ter uma superioridade ao SQL Server 2012 em quase todas as operações, porém, no quesito Taxa de Erro o MongoDB foi 66,6% mais errôneo que o SGBD relacional. Com isso, pode-se concluir que a velocidade superior na execução das operações é devida a taxa de erro também superior.

Ainda no quesito Taxa de Erro, é notável o comportamento do MongoDB quanto ao número de operações ao qual ele é submetido, uma vez que, nas etapas

de 1.000 execuções existe uma taxa de erro X e, quando a quantidade de operações submetidas à ele é aumentada para 10.000, a taxa de erro também tende a aumentar. Mas, quando a quantidade de operações é aumentada para 100.000, a taxa de erro tende a cair novamente. Já o SQL Server 2012 tende a aumentar a taxa de erro conforme a quantidade de operações também é aumentada.

Desta vez no quesito *throughput*, o MongoDB apresentou uma taxa maior que a do SQL Server 2012 na maioria dos casos e, isto também está relacionado diretamente a taxa de erro de ambos. Como pode ser visto, o MongoDB sempre manteve uma taxa de *throughput* e taxa de erro estáveis, mas, no caso do SQL Server 2012 a taxa de *throughput* diminui conforme a taxa de erro aumenta e vice-versa.

Com essas informações chega-se a conclusão de que para aplicações em que não seja necessária concorrência, ambos os SGBDs em suas configurações iniciais atendem perfeitamente aos requisitos. Porém, é notável que a tecnologia NoSQL demonstra uma vantagem em relação a um SGBD relacional.

No caso de aplicações em que é necessária uma grande concorrência em manipulação de informações, o SGBD relacional em suas configurações iniciais demonstrou um melhor desempenho que a tecnologia NoSQL.

Portanto, é visível que a tecnologia NoSQL é superior a tecnologia do Modelo Relacional, por exemplo, em aplicações de relatórios. Onde na maioria dos casos são executados de forma não concorrente e é necessária a manipulação de um grande volume de informações de maneira rápida, com um menor consumo de memória.

Porém, em aplicações comerciais onde, na maioria dos casos é necessária uma grande concorrência pelo uso das informações, a tecnologia do Modelo Relacional ainda apresenta uma vantagem em relação a tecnologia NoSQL

5.2 PERSPECTIVAS FUTURAS

Um dos únicos inconvenientes encontrados durante o desenvolvimento deste trabalho foi a escassez de ferramentas não proprietárias (software livre) para

os testes, uma vez que, somente a Apache oferece a ferramenta JMeter que tem recursos para testes diretamente em SGBDs fazendo uso de *plugins.*

Outras ferramentas encontradas são somente para testes de requisições HTTP, sendo necessário o desenvolvimento de arquivos que se comunicam com um SGBD através de uma linguagem de programação que seja executada por um servidor WEB.

Diante deste cenário, um trabalho futuro interessante é o desenvolvimento de uma nova ferramenta de testes para SGBDs relacionais e NoSQL através de drivers de conexão JDBC.

REFERÊNCIAS

10GEN.**Driver JDBC MongoDB**. Disponível em: <http://central.maven.org/maven2/org/mongodb/mongo-java-driver/2.9.3/>. Acesso em 30 julho de 2013.

10GEN.**MongoDB**. Disponível em: <http://www.mongodb.org/downloads>. Acesso em 10 julho de 2013.

APACHE FOUNDATION.**Apache JMeter**. Disponível em: <https://jmeter.apache.org/download_jmeter.cgi>. Acesso em 30 julho de 2013.

CHODOROW, K.**MongoDB:** The DefinitiveGuide.2.Ed. Sebastopol, CA: O'Reilly Media, 2013.

ISSA, F. G. S. **Estudo comparativo entre Banco de Dados Relacionais e Banco de Dados NoSQL na Utilização por Aplicações de Business Intelligence**. 2011. Trabalho de Conclusão de Curso. (Graduação em Banco de Dados) - Fatec São José dos Campos. Orientador: Fernando Masanori Ashikaga.

MICROSOFT CORPORATION.**Driver JDBC SQL Server**. Disponível em: <http://www.microsoft.com/en-s/download/details.aspx?id=11774>. Acesso em 30 julho de 2013.

MICROSOFT CORPORATION.**SQL Server 2012 Enterprise Edition**. Disponível em: <http://www.microsoft.com/en-us/sqlserver/editions/2012-editions/enterprise.aspx>. Acesso em 30 julho de 2013.

SANTOS, I. S.; NETO, P. A. S. **Automação de Testes de Desempenho e Estresse com Jmeter**. Disponível em: <http://www.ufpi.br/subsiteFiles/pasn/arquivos/files/artigoJMeter(1).pdf>. Acesso em 12 de setembro de 2013.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. Trad. Sob a direção de Daniel Vieira, Rio de Janeiro, Elsevier, 2006 – 3ª reimpressão.

GLOSSÁRIO DE TERMOS TÉCNICOS

Banco de Dados	Agrupamento de unidades de armazenamento (tabelas, coleções) que podem ou não estar inter-relacionadas.
BI	Refere-se à Inteligência Empresarial (método de obtenção de informações gerenciais por meio de análise inteligente de dados).
C++	Linguagem de programação compilada e orientada a objetos utilizada em ambiente desktop e sistemas embarcados.
Coleção	A menor unidade de armazenamento do sistema gerenciador de bancos de dados MongoDB.
Coluna	A menor unidade de armazenamento de um sistema gerenciador de bancos de dados relacionais.
DVD	Mídia de armazenamento de dados que pode ser utilizada como memória secundária em sistemas computacionais.
Esquema	Coleção de objetos de um banco de dados que estão disponíveis à um usuário ou grupo (tabelas, visões, funções, etc.).
GridFS	Método de armazenamento de dados utilizado pelo MongoDB.
Hardware	Diz respeito à um componente físico de um sistema computacional (placa, disco, etc.).
HD	Memória secundária de um sistema computacional.
Java	Linguagem de programação compilada e interpretada, orientada a objetos e multiplataforma utilizada tanto em ambiente desktop, quanto no ambiente WEB.
JavaScript	Linguagem de programação interpretada e muito difundida no ambiente WEB.
Plugin	Módulo de extensão de um programa de computador utilizado para adicionar funcionalidades a outros programas maiores.
RAM	Memória primária de um sistema computacional.
SQL	Linguagem de consulta utilizada por sistemas gerenciadores de bancos de dados para a criação e manipulação de esquemas.
Software	Diz respeito a um componente não físico de um sistema

	computacional (programa, aplicativo, utilitário, etc.).
TAG	Palavra-chave que se refere à, e identifica determinado dado em um documento de notação.
Throughput	Medida da transferência de dados em um determinado circuito durante um determinado período de tempo.
Vazão	Quantidade de ações ou transações que podem ser realizadas por um sistema durante um determinado período de tempo.
WEB	Ambiente virtual referente à grande rede de computadores interligados por meio da internet.

APÊNDICE A – Scripts de Criação de Tabelas e Procedimentos do SGBD MS SQL Server 2012

```
CREATE TABLE dbo.itens (
	id INT NOT NULL IDENTITY(1,1),
	nome VARCHAR(32) NOT NULL,
	descricao VARCHAR(500) NOT NULL,
	valor DECIMAL(10, 2) NOT NULL,
	PRIMARY KEY(id)
);

CREATE TABLE dbo.vendas (
	id INT NOT NULL IDENTITY(1,1),
	dt_venda DATETIME NOT NULL,
	total DECIMAL(10, 2) NOT NULL,
	valor_pago DECIMAL(10, 2) NOT NULL,
	troco DECIMAL(10, 2) NOT NULL,
	PRIMARY KEY(id)
);

CREATE TABLE dbo.itens_vendidos (
	id_venda INT NOT NULL,
	id_item INT NOT NULL,
	quantidade DECIMAL(10, 3) NOT NULL,
	subtotal DECIMAL(10, 2) NOT NULL,
	PRIMARY KEY(id_venda, id_item),
	CONSTRAINT itens_vendidos_FK1
		FOREIGN KEY(id_venda)
			REFERENCES dbo.vendas(id),
	CONSTRAINT itens_vendidos_FK2
		FOREIGN KEY(id_item)
			REFERENCES dbo.itens(id)
);
```

```
CREATE PROCEDURE transaction_insert AS

        DECLARE @id_item INT;

        DECLARE @id_venda INT;

        BEGIN TRANSACTION;

        INSERT INTO itens (nome, descricao, valor) VALUES ('Salgado', 'Salgado
frito', 2.56);

        SELECT @id_item = IDENT_CURRENT('itens');

        INSERT INTO vendas (dt_venda, total, valor_pago, troco) VALUES
(CURRENT_TIMESTAMP, 7.78, 8.0, 0.22);

        SELECT @id_venda = IDENT_CURRENT('vendas');

        INSERT INTO itens_vendidos (id_venda, id_item, quantidade, subtotal)
VALUES (@id_venda, @id_item, 3, 7.68);

        COMMIT TRANSACTION;


CREATE PROCEDURE transaction_update @id1 INT, @id2 INT AS

        BEGIN TRANSACTION;

        UPDATE itens SET nome = 'Coca-cola', descricao = 'Refrigerante de Cola',
valor = 3.00 WHERE id = @id1;

        UPDATE vendas SET dt_venda = CURRENT_TIMESTAMP, total = 6.00,
valor_pago = 6.00, troco = 0.00 WHERE id = @id2;

        UPDATE itens_vendidos SET quantidade = 2.00, subtotal = 6.00 WHERE
id_venda = @id2 AND id_item = @id1;

        COMMIT TRANSACTION;


CREATE PROCEDURE transaction_delete @id1 INT, @id2 INT AS

        BEGIN TRANSACTION;

        DELETE FROM itens_vendidos WHERE id_venda = @id2 AND id_item =
@id1;

        DELETE FROM itens WHERE id = @id1;

        DELETE FROM vendas WHERE id = @id2;

        COMMIT TRANSACTION;
```

```
CREATE PROCEDURE transaction_list AS
        SELECT * FROM itens AS i
        INNER JOIN itens_vendidos AS iv ON i.id = iv.id_item
        INNER JOIN vendas AS v ON v.id = iv.id_venda;
```

APÊNDICE B – Scripts de Consulta do SGBD MongoDB

```
db.transacoes.insert({
        "id": 1,
        "nome":"coca-cola",
        "descricao":"refrigerante de cola",
        "valor":3.45,
        "dt_venda":"2013-11-25",
        "quantidade":3,"total":10.35,
        "valor_pago":11,"troco":0.65
})

db.transacoes.update({
        "nome":"coca-cola"
},
{
        $set:{
                "id": 1,
                "nome":"salgado",
                "descricao":"salgado frito",
                "valor":2.56,
                "dt_venda":"2013-11-25",
                "quantidade":4,
                "total":10.24,
                "valor_pago":11,
                "troco":0.76
        }
},
{
        upsert:false,
        multi:false
})
```

```
db.transacoes.remove({
        "nome":"salgado"
},1)

db.transacoes.find()
```

APÊNDICE C – Configuração das Conexões e dos Testes no Apache JMeter

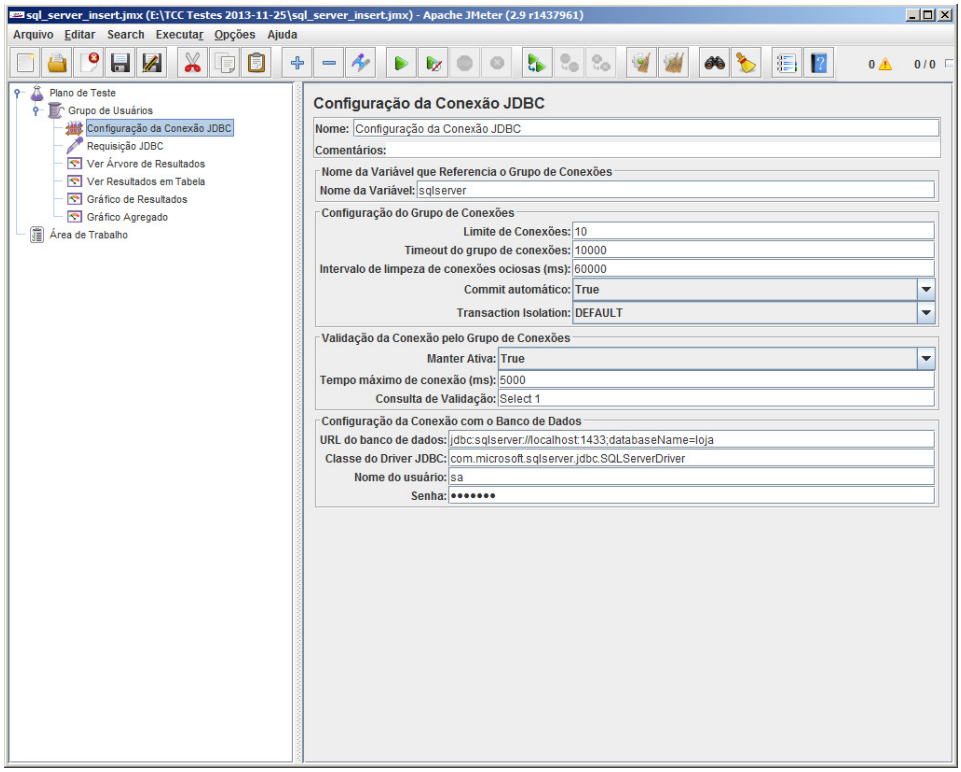


Figura 50: Configuração da Conexão JDBC com o SQL Server.

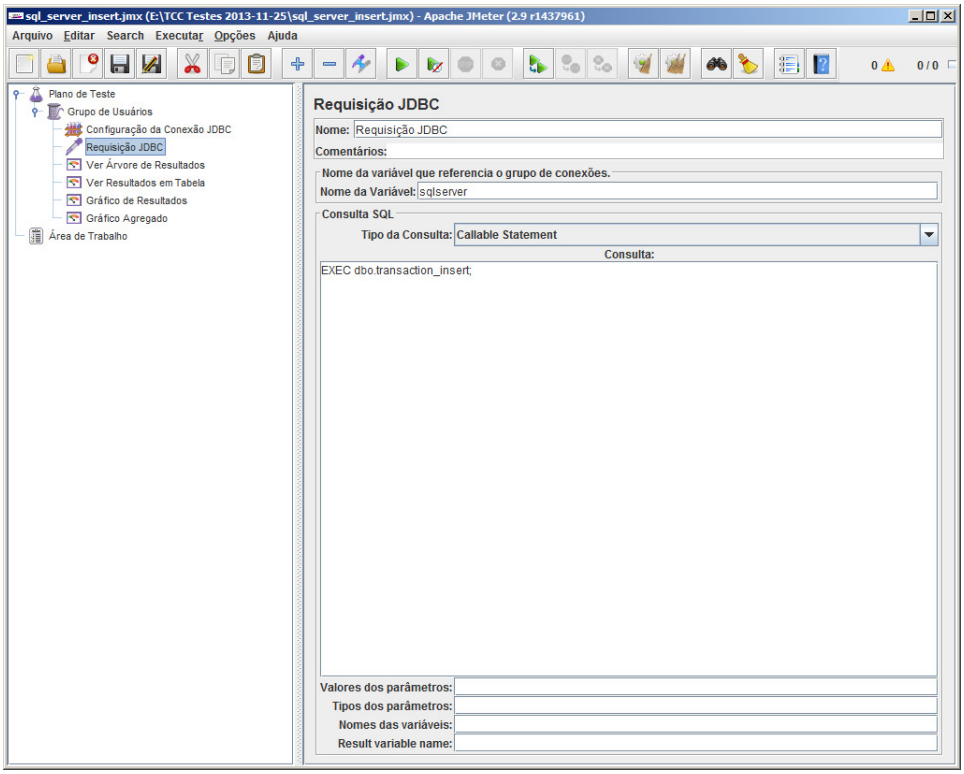


Figura 51: Configuração da requisição de *INSERT* do SQL Server.

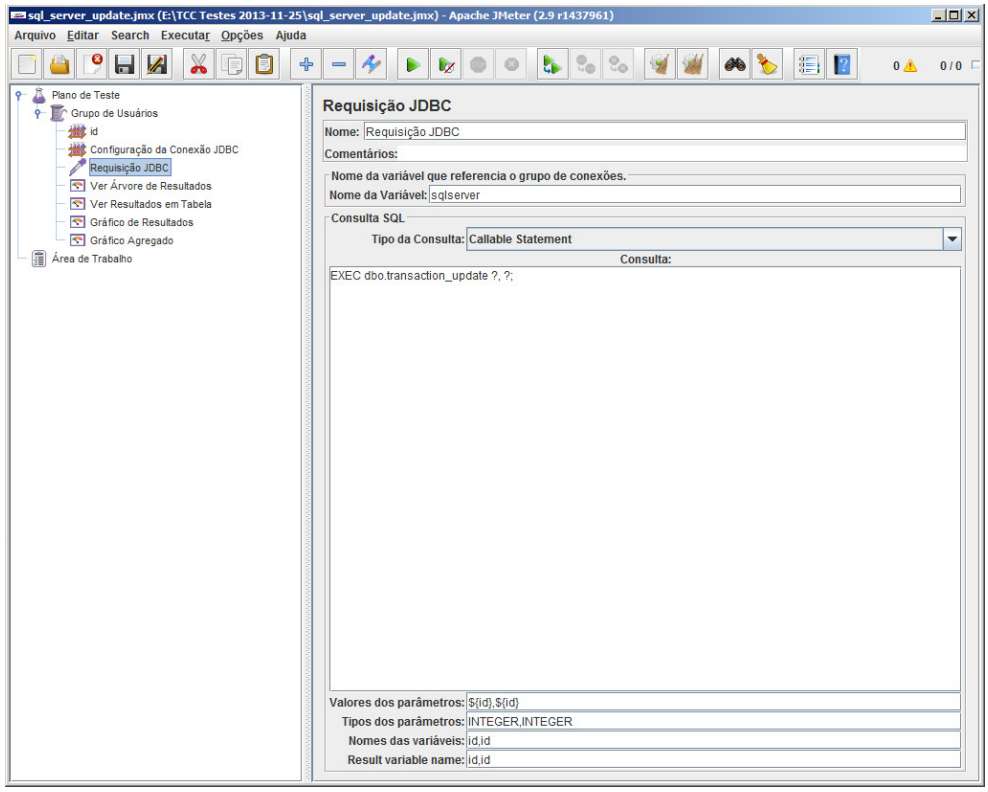


Figura 52: Configuração da requisição de *UPDATE* do SQL Server.

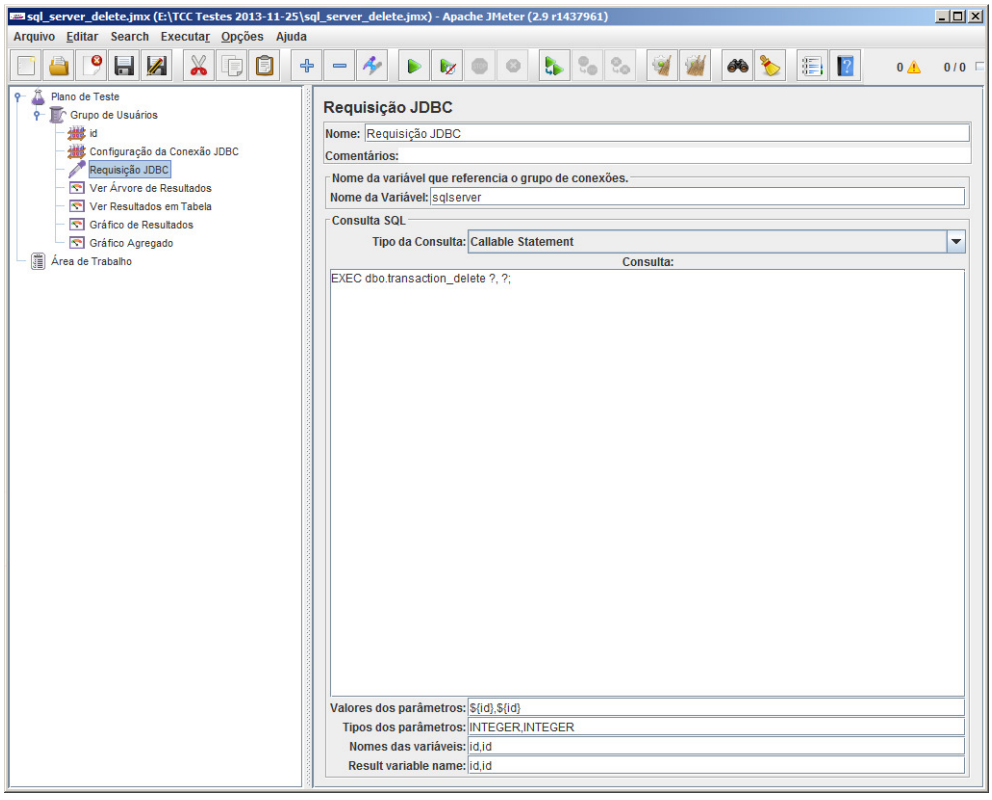


Figura 53: Configuração da requisição de *DELETE* do SQL Server.

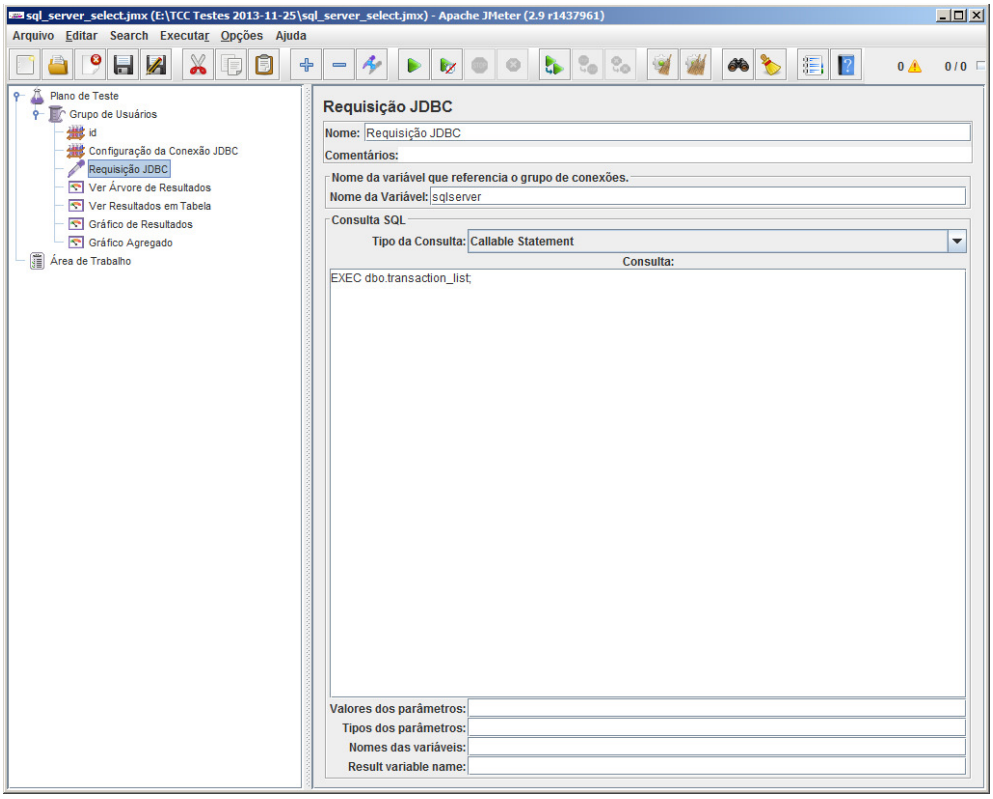


Figura 54: Configuração da requisição de *SELECT* do SQL Server.

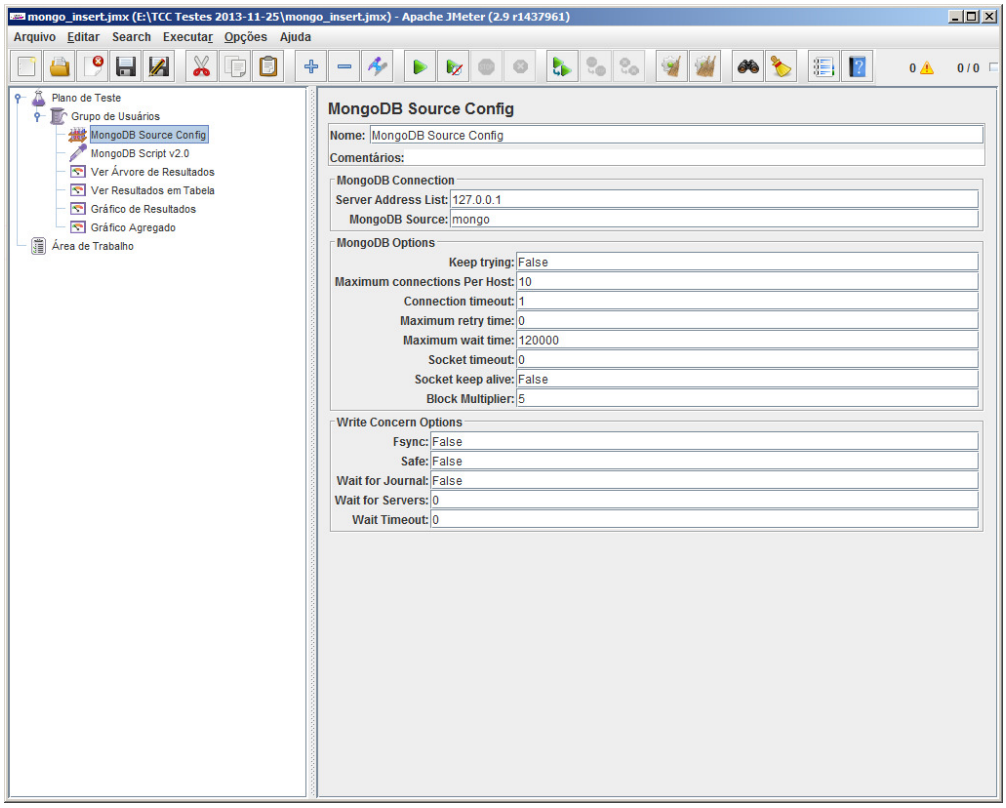


Figura 55: Configuração da Conexão JDBC com o MongoDB.

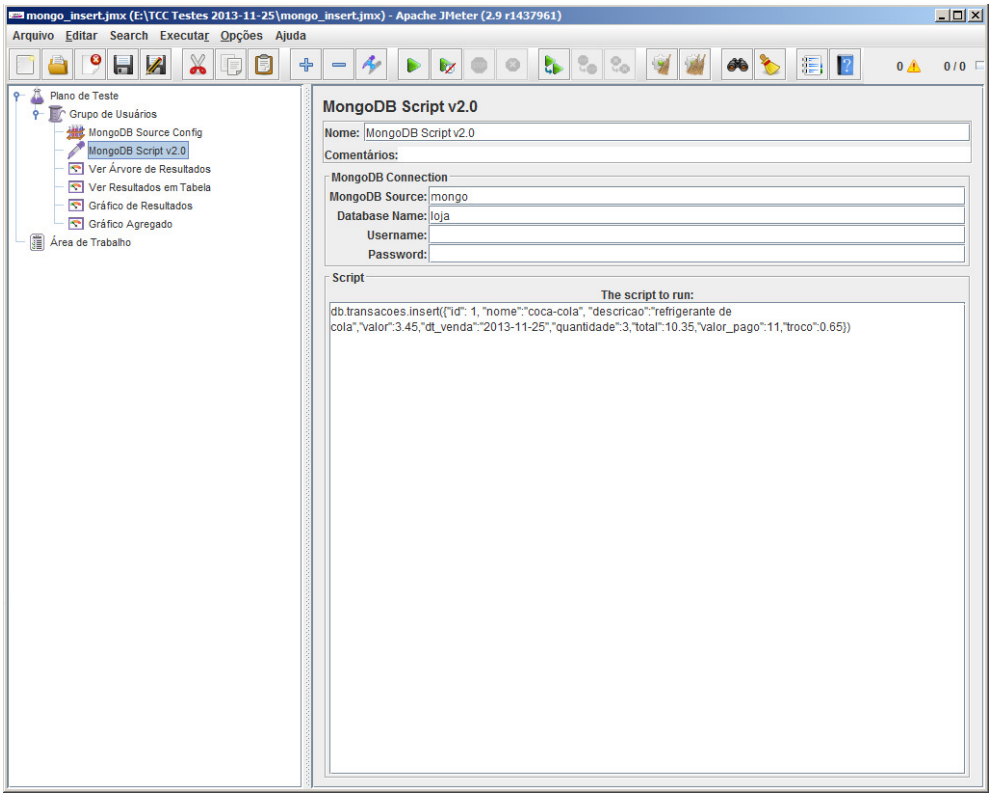


Figura 56: Configuração da requisição de *INSERT* do MongoDB.

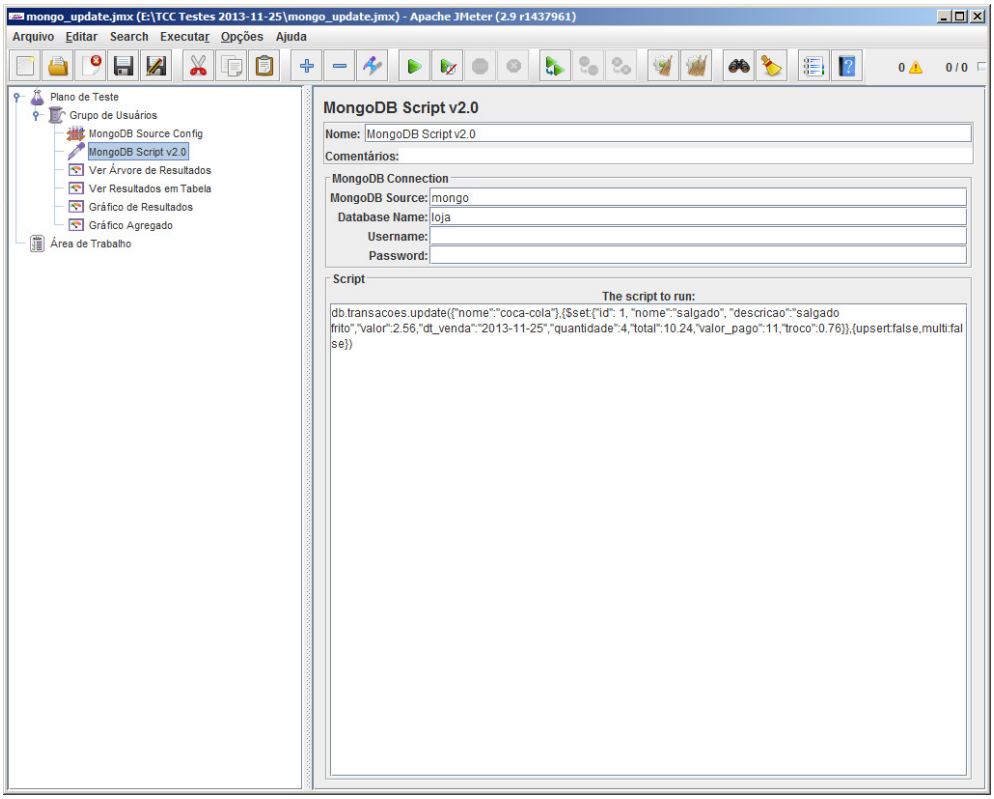


Figura 57: Configuração da requisição de *UPDATE* do MongoDB.

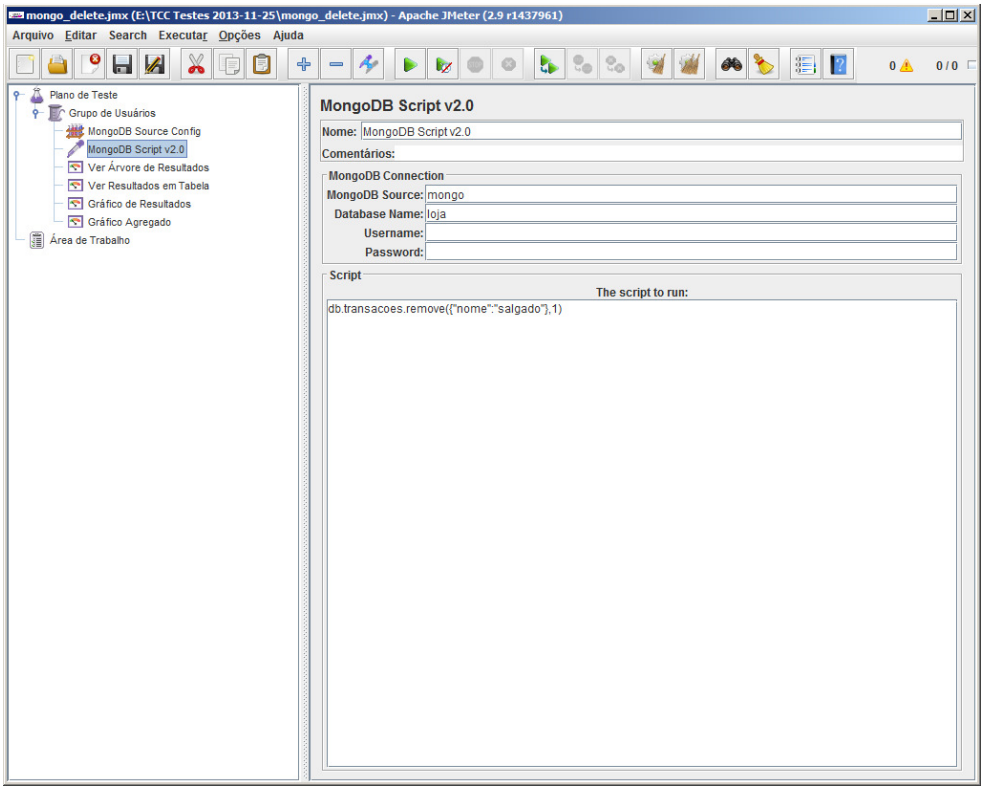


Figura 58: Configuração da requisição de *REMOVE* do MongoDB.

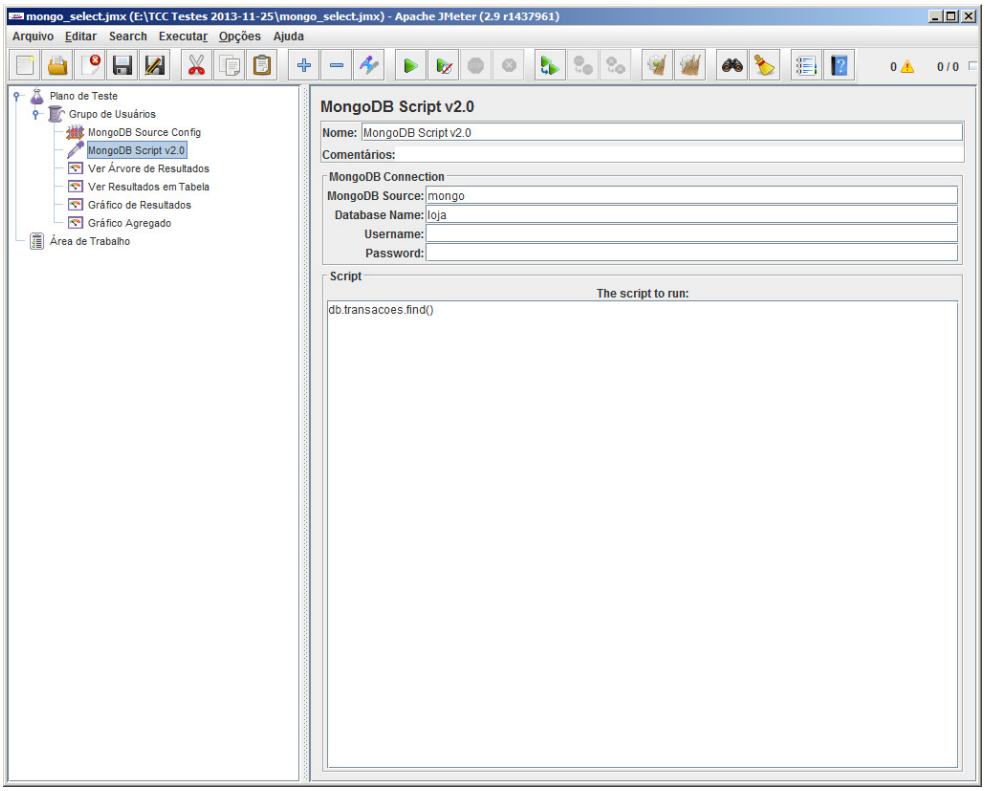


Figura 59: Configuração da requisição de *FIND* do MongoDB.