Instituto Superior de Engenharia do Porto

ARMAZÉNS DE DADOS EM BASES DE DADOS NOSQL

Daniel José Pinto Pereira

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Tecnologias do conhecimento e decisão

Orientador: Doutor Paulo Oliveira, DEI/ISEP

Júri:

Presidente:

Doutora Maria de Fátima Rodrigues, DEI/ISEP

Vogais:

Doutor Paulo Maio, DEI/ISEP

Doutor Paulo Oliveira, DEI/ISEP

Porto, Outubro de 2014

"Não sei por onde vou,
Não sei para onde vou
Sei que não vou por aí!"

José Régio, in 'Poemas
de Deus e do Diabo'

Resumo

Na atualidade, existe uma quantidade de dados criados diariamente que ultrapassam em muito as mais otimistas espectativas estabelecidas na década anterior. Estes dados têm origens bastante diversas e apresentam-se sobre várias formas. Este novo conceito que dá pelo nome de Big Data está a colocar novos e rebuscados desafios ao seu armazenamento, tratamento e manipulação. Os tradicionais sistemas de armazenamento não se apresentam como a solução indicada para este problema. Estes desafios são alguns dos mais analisados e dissertados temas informáticos do momento. Várias tecnologias têm emergido com esta nova era, das quais se salienta um novo paradigma de armazenamento, o movimento NoSQL. Esta nova filosofia de armazenamento visa responder às necessidades de armazenamento e processamento destes volumosos e heterogéneos dados.

Os armazéns de dados são um dos componentes mais importantes do âmbito *Business Intelligence* e são, maioritariamente, utilizados como uma ferramenta de apoio aos processos de tomada decisão, levados a cabo no dia-a-dia de uma organização. A sua componente histórica implica que grandes volumes de dados sejam armazenados, tratados e analisados tendo por base os seus repositórios. Algumas organizações começam a ter problemas para gerir e armazenar estes grandes volumes de informação. Esse facto deve-se, em grande parte, à estrutura de armazenamento que lhes serve de base. Os sistemas de gestão de bases de dados relacionais são, há algumas décadas, considerados como o método primordial de armazenamento de informação num armazém de dados. De facto, estes sistemas começam a não se mostrar capazes de armazenar e gerir os dados operacionais das organizações, sendo consequentemente cada vez menos recomendada a sua utilização em armazéns de dados.

É intrinsecamente interessante o pensamento de que as bases de dados relacionais começam a perder a luta contra o volume de dados, numa altura em que um novo paradigma de armazenamento surge, exatamente com o intuito de dominar o grande volume inerente aos dados *Big Data*. Ainda é mais interessante o pensamento de que, possivelmente, estes novos sistemas NoSQL podem trazer vantagens para o mundo dos armazéns de dados. Assim, neste trabalho de mestrado, irá ser estudada a viabilidade e as implicações da adoção de bases de dados NoSQL, no contexto de armazéns de dados, em comparação com a abordagem tradicional, implementada sobre sistemas relacionais. Para alcançar esta tarefa, vários estudos foram operados tendo por base o sistema relacional SQL Server 2014 e os sistemas NoSQL, MongoDB e Cassandra. Várias etapas do processo de desenho e implementação de um armazém de dados foram comparadas entre os três sistemas, sendo que três armazéns de dados distintos foram criados tendo por base cada um dos sistemas. Toda a investigação realizada neste trabalho culmina no confronto da performance de consultas, realizadas nos três sistemas.

Palavras-chave: *NoSQL*, *Big Data*, Armazéns de dados, SQL Server 2014, MongoDB, Cassandra, Business Intelligence

Abstract

Nowadays, the amount of daily created data goes far beyond the most optimistic expectations, established in the previous decades. This data has very different backgrounds and is presented in several forms. This new concept goes by the name of Big Data, and is creating new and convoluted challenges to its storage, processing and handling. Traditional storage systems do not arise as the right solution for this problem. These challenges are some of the most analyzed and studied informatics topics of the moment. Several technologies have emerged with this new era, from which stands out a new storage paradigm, the NoSQL movement. This new philosophy aims to answer to the storage and processing needs of these heterogeneous and voluminous data.

Data warehouses are one of the major component in the Business Intelligence context and are mostly used as a tool to support the decision-making process carried out daily in an organization. Its historical component implies that large amounts of data are stored, processed and analyzed based on their repositories. Some organizations are starting to have problems to manage and store these large volumes of information. This is, in large part, due to the storage structure on which they are based. For some decades now, the relational database management systems have been considered as the primary method for storing information in a data warehouse. Actually these systems are starting to not be able of storing and managing operational data from organizations, their use in data warehouse is consequently becoming less recommended.

It is intrinsically interesting that relational databases are starting to lose the fight against the data volume, at a time when a new storage paradigm emerges, precisely aiming to dominate the large data volume inherent to the Big Data era. Even more interesting is the idea that, possibly, these new NoSQL systems can bring benefits to the world of data warehouses. Thus, in this dissertation, will be studied the feasibility and implications of the adoption of NoSQL databases in the context of data warehouses, in comparison with the traditional approach, implemented on relational systems. To achieve this task a number of studies have been operated based on the relational system, SQL Server 2014 and on the NoSQL systems, MongoDB and Cassandra. Various stages of the process of designing and implementing a data warehouse were compared among the three systems, which lead to the creation of three distinct data warehouses, one based on each system. All research carried out in this work culminates in the performance comparison of queries held in the three systems.

Keywords: *NoSQL*, *Big Data*, Data Warehousing, SQL Server 2014, MongoDB, Cassandra, Business Intelligence

Agradecimentos

A realização deste trabalho não teria sido possível sem o contributo de várias pessoas. Assim, gostaria de agradecer a todos os que de algum modo contribuíram para a realização deste trabalho de mestrado de uma forma direta ou indireta. Salientado um agradecimento especial:

Aos meus pais, José Pereira e Fátima Pinto, pelo acompanhamento vitalício, pelo apoio inabalável, pela eterna paciência, pelo longo patrocínio nesta e noutras aventuras e por todos os sacrifícios ultrapassados para que todos os meus sonhos estivessem ao alcance.

À Maria João Pereira e ao Pedro Melo, por me deixaram escrever nos seus computadores e por me convencerem a participar em atividades recreativas quando deveria estar a trabalhar neste documento.

Ao Marcelo Silva, pela calculadora emprestada e pela paciência que me vai gastando ao longo dos anos.

Ao Alexis Rodrigues, Tiago "Belmiro" Pereira, Dino da Costa, João Lopes, João Oliveira, Marco Pinto, Ruben Oliveira e Vítor Neto, por todos os grandes momentos, pela ajuda na finalização da licenciatura e ao mesmo tempo pela ajuda no atraso na finalização da licenciatura.

Ao Bruno Soares e ao Diogo Garcia pela ajuda na integração, pela dedicação empregue nos vários trabalhos realizados ao longo deste mestrado, e por serem uns excelentes amigos ao longo desta segunda caminhada académica.

Ao Professor Doutor Paulo Oliveira pela orientação, pela revisão do trabalho, pela partilha de conhecimento, pela disponibilidade, pela enorme paciência e visão, por responder sempre a todas as mensagens de tamanho bíblico que lhe eram endereçadas, por conceber o tema desta dissertação e por todo o esforço que colocou neste trabalho.

À pequena mas muito importante Inês Ferreira, pela enorme motivação, pela constante dedicação, pela demasiado longa paciência, por ser a melhor revisora ortográfica, por ter contribuído bastante para a conclusão deste documento, por ter contribuído bastante para o atraso na conclusão deste documento e por todos os Sábados e Domingos em que ficou muito chateada por este trabalho de mestrado ter de ser desenvolvido.

A todos os meus familiares, amigos, professores e colegas que, de algum modo, me ajudaram a concluir a realização deste trabalho.

A todos, o meu sincero obrigado…

Índice

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Acrónimos e Símbolos

Lista de Acrónimos

ACID	*Atomicity, Consistency, Isolation, Durability*
API	*Application Programming Interface*
AWS	*Amazon Web Services*
BASE	*Basically Available*, *Soft state*, *Eventual consistency*
BI	*Business Intelligence*
BSON	*Binary JSON*
CAP	*Consistency, Availability, Partition tolerance*
CDR	*Call Detail Record*
CERN	*Conseil Européen pour la Recherche Nucléaire*
CLI	*Command-Line Interface*
CPU	*Central Processing Unit*
CQL	*Cassandra Query Language*
CRM	*Customer Relationship Management*
ERP	*Enterprise Resource planning*
GFS	*Google File System*
HDFS	*Hadoop Distributed File System*
HiveQL	*Hive Query Language*
HQL	*Hypertable Query Language*
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
LINQ	*Language-INtegrated Query*
M2M	*Machine to Machine*

Lista de Acrónimos (continuação)

MAD *Magnetism*, *Agility*, Depth

MMS *Multimedia Messaging Service*

MPP *Massively Parallel Processing*

MVCC *MultiVersion Concurrency Control*

NoSQL *Not Only SQL*

OLAP *OnLine Analytical Processing*

PDF *Portable Document Format*

RAM *Random Access Memory*

REST *REpresentational State Transfer*

RFID *Radio-Frequency IDentification*

SMS *Short Message Service*

SQL *Structured Query Language*

TPC *Transactional Processing Performance Council*

URI *Uniform Resource Identifier*

URL *Uniform Resource Locator*

XML *eXtensible Markup Language*

1 Introdução

Este capítulo tem como finalidade fornecer ao leitor todas as informações necessárias para que se possa enquadrar plenamente no âmbito e na filosofia da investigação realizada. Serão elucidados outros pontos relevantes para a auto avaliação e validação do trabalho desenvolvido, como os objetivos a alcançar e as contribuições que se esperam obter. Ao mesmo tempo irá também o leitor ser contextualizado em questões, também importantes mas tangíveis, como as motivações pessoais e académicas que sustentam este projeto investigativo. Por fim, irá ser fornecida uma visão sobre a estrutura deste documento e o seu significado.

1.1 Contextualização: História, Dados e a Cultura Digital

Dados… Os mais pequenos grãos de areia nesta praia, cada vez mais digital, a que se chama de vida no século *XXI*. Organizações, pessoas e objetos veem-se cada vez mais interligados e dependentes de uma rede digital internacional, onde todas as palavras, gestos, ações e movimentos resultam num bloco de dados. Em grande parte, vivemos cega e voluntariamente na matriz professada pelos irmãos *Wachowski* [1]. Nada neste avançado mundo é desperdiçado, os nossos resíduos digitais são liquidificados em dados que nos são restituídos através de ações de marketing personalizadas, ao qual a nossa resposta (ou falta dela) despoletará ações que resultarão em mais dados, e o ciclo recomeça. Os dados são a poeira estelar deste século que ainda é escrito.

Nós, humanos, somos há muito atraídos pelo poder da informação confinada aos dados crus e inexplorados. Este fascínio data de anos ainda anteriores à instituição da civilização grega, esta ligação remonta aos inícios do comércio e da inerente contabilidade (Krishnan, 2013). O comércio e a ciência têm sido ao longo do tempo os grandes impulsionadores dos processos de extração de conhecimento de dados. Com a massificação da utilização de sistemas de informação por parte das organizações, tornou-se claro que existia um potencial ínfinito a tirar

[1] *Lana* e *Andrew Wachowski*, são dois irmãos, realizadores e argumentistas norte-americanos, especialmente reconhecidos pelo seu filme de 1999, “The Matrix”.

dos dados operacionais resultantes das atividades do dia-a-dia de uma organização. No final da década de 80 do século passado, surge o conceito de *Business Intelligence* (Lim et al., 2013). Desde então vários métodos, técnicas e ferramentas têm sido albergados debaixo deste termo. Tal como físicos e astrónomos, que buscam respostas para o futuro no passado do cosmos, as organizações descobriram que as informações escondidas nos dados históricos lhes poderiam trazer um maior conhecimento do futuro. Assim, os armazéns de dados surgem como resposta à necessidade de tratar, armazenar e analisar todos os dados "históricos" de uma organização (Kimball e Ross, 2013). Os armazéns de dados são considerados uma tecnologia nuclear no âmbito *Business Intelligence*. Devido à exponencial necessidade de informação, cada vez mais profunda, concisa e rápida, os sistemas *BI*, em geral, e os armazéns de dados, em particular, foram gradualmente evoluindo ao longo dos últimos 20 anos. Apesar desta evolução, os armazéns de dados ainda hoje possuem no seu núcleo estruturas que seguem um paradigma estabelecido nos anos 70 do século *XX*: bases de dados relacionais. Na atualidade existem organizações com necessidades de armazenar mais de 100 *terabytes* de dados nos seus armazéns de dados (Imhoff et al., 2003). Conseguir armazenar este grande volume de dados de uma forma eficiente e, ao mesmo tempo ser capaz de fornecer análises rápidas e multidimensionais tem-se tornado um desafio cada vez maior para empresas de *software* e orçamentos de organizações. O paradigma e a arquitetura relacional, começam cada vez mais a ceder perante o volume de dados e as caraterísticas das análises requeridas (Stonebraker et al., 2007) (Stonebraker e Cetintemel, 2005). O mundo digital mudou com o novo século. Os desafios para os armazéns de dados relacionais tornam-se cada vez maiores porque um novo paradigma emerge…

Big Data… Um termo de significado abstrato, que se tornou na palavra quente do mundo informático desde que foi pela primeira vez aplicada, algures na segunda metade da primeira década deste século. O que é realmente *Big Data*? Dados, dados, dados… O apogeu das empresas e serviços *web*, conjugado com a massificação de dispositivos inteligentes e da internet móvel levaram a uma explosão no volume de dados, e a previsão é que este continue a aumentar. O primeiro parágrafo deste subponto, para além de uma caricatura dramática da cultura digital em que vivemos, é também a confirmação de que vivemos cada vez mais rodeados de dados. O potencial conhecimento que se pode obter da análise destes dados é inimaginável. Como tal, empresas e académicos dedicam-se a estudar e experimentar este novo paradigma. Como é que, no entanto, num dia de chuva continua a cair água das nuvens e não dados? Que paradigma é este que permite à nuvem armazenar um tamanho tão volumoso de dados, muito para além do potencial de um armazém de dados? *NoSQL*… será isto um paradigma ou um movimento do contra? Será que os armazéns de dados, inatamente relacionais, podem tirar vantagem deste novo paradigma de armazenamento?

1.2 Formulação do caso de estudo

O subponto anterior estabelece o cenário de onde se retirou o tema desta investigação. Uma leitura cuidada leva a que sejam salientáveis várias questões dignas de serem dissertadas. Neste subponto será elucidado qual é o tema a investigar com este projeto, e por que motivo se salientou de entre muitos outros presentes no cenário apresentado. A importância dos armazéns de dados para a sociedade moderna é indiscutível. Do cenário anterior é possível concluir, no entanto, que as bases de dados relacionais presentes nos armazéns de dados tradicionais começam a claudicar com a pressão exercida pelo tamanho dos dados armazenados. As questões deixadas no ar no subponto anterior podem resumir-se numa única questão: se o paradigma de armazenamento *NoSQL* tem por objetivo suportar grandes volumes de dados, será que é possível que sejam utilizados para suprimir o grande problema de armazenamento que se avizinha, nos armazéns de dados relacionais?

Assim formou-se a questão principal que esta dissertação pretende investigar: Qual a viabilidade da utilização de bases de dados *NoSQL*, como ferramenta de armazenamento num armazém de dados?

A realidade é que temas como: *business intelligence*, armazéns de dados e bases de dados relacionais têm desde há décadas sido utilizados como tema de investigações, estudos e dissertações. Apesar de ainda existirem bastantes pontos dignos de serem estudados, estes temas, de um ponto de vista pessoal, já não se revelam extremamente interessantes, motivadores e divertidos para conseguirem sustentar todo o processo de desenvolvimento de um projeto investigativo desta magnitude. Assim decidiu-se orientar esta dissertação para um prisma Big Data e NoSQL, mas mantendo a questão principal já definida.

1.3 Objetivos e Contribuições esperadas

Como os temas *Big Data* e *NoSQL* são relativamente recentes, pretende-se primeiramente contribuir para uma maior introspeção e esclarecimento sobre o que são estes conceitos e qual o seu papel no mundo digital em que vivemos. Para se alcançar esse efeito definiram-se os seguintes objetivos:

- Identificação dos conceitos relacionados com *Big Data*/bases de dados *NoSQL*
- Levantamento dos principais paradigmas/modelos de bases de dados *NoSQL*

Com este estudo pretende-se concluir se é realmente viável recorrer a bases de dados *NoSQL*, para servir como sistema de armazenamento de armazéns de dados. Assim, espera-se contribuir para um melhor conhecimento dos processos, dificuldades e diferenças que existem entre o processo de implementação de um armazém de dados num sistema *NoSQL* e num sistema relacional. Este estudo só poderia ser corretamente realizado com a implementação real, numa máquina, de um armazém de dados em sistemas *NoSQL*. É impossível ter uma

verdadeira noção dos processos que são necessários para realizar esta tarefa se este estudo se mantiver apenas num plano teórico. Desta forma, espera-se conseguir desenvolver um protótipo de um armazém de dados em sistemas *NoSQL* capaz de responder a várias consultas. Também se pretende que este estudo seja particularmente útil para investigadores e organizações que, no futuro, considerem esta mudança de paradigma de armazenamento. Para tal definiram-se os seguintes objetivos a alcançar:

- Estudo comparativo das diferentes ferramentas de bases de dados *NoSQL*
- Análise comparativa da implementação de um armazém de dados numa base de dados relacional e em bases de dados *NoSQL*

Para que um estudo sobre viabilidade da utilização de bases de dados *NoSQL* em armazéns de dados consiga realmente contribuir para um melhor conhecimento é necessário avaliar como estas bases de dados se comportam ao responderem a consultas realizadas sobre os dados. A performance de consultas é habitualmente um ponto crítico no desenho de um armazém de dados e, por esse motivo, este fator será utilizado como o componente a avaliar. Para se conseguir alcançar estes pontos, definiu-se o seguinte objetivo:

- Estudo comparativo do desempenho/performance das análises de dados realizadas sobre um armazém de dados implementado numa base de dados relacional e em bases de dados *NoSQL*

Uma dissertação é um processo bastante dinâmico em que novos conhecimentos e curiosidades académicas vão surgindo com naturalidade ao longo do processo de desenvolvimento. Assim, alguns objetivos secundários foram sendo definidos ao longo da investigação e os mesmos serão referidos e elucidados ao longo deste documento. Um destes objetivos secundários é conseguir compreender qual a relação entre armazéns de dados e Big Data, e quais as mudanças que têm de ser levadas a cabo para se conseguir integrar e analisar dados Big Data contidos num armazém de dados.

1.4 Motivação

O conceito *Big Data* tem sido um dos mais discutidos e analisados nos últimos anos. Esta palavra tornou-se no epicentro de um variado número de novas tecnologias e conceitos. É, neste momento, praticamente impossível estar a par de todos os desenvolvimentos neste âmbito pois novas tecnologias, estudos e ideias parecem surgir à velocidade da luz. O que hoje é aceite como uma certeza, amanha não passa de uma teoria. Este novo paradigma parece ter captado não só a atenção do mundo informático, como também a do mundo empresarial. As organizações só começam agora a perceber a importância e o impacto destes dados. Ainda faltam bastantes anos até que médias empresas consigam tirar proveito destas tecnologias.

Cada vez mais os “dados” são apontados como o novo ouro. Sendo *Big Data* considerado como o futuro dos dados, é possível chegar à conclusão que esta dissertação não poderia estar em maior sintonia com a atualidade. Este projeto representa uma oportunidade única para se dar

um primeiro passo em relação à compreensão deste mundo e à especialização nas suas tecnologias, como as bases de dados *NoSQL*. Por outro lado, os armazéns de dados têm sido o braço direito das organizações, na conquista de conhecimento e introspeção. A sua importância na sociedade empresarial é inegável, e o seu peso nos catálogos das empresas de *software* também. Novos desenvolvimentos são eminentes neste âmbito.

Os temas a abordar nesta dissertação são atuais, interessantes e apresentam um vislumbre do futuro. As motivações pessoais e académicas para este projeto não poderiam ser maiores.

1.5 Organização do Documento

Nesta subsecção descreve-se a organização deste documento e sintetizam-se cada um dos oito capítulos que constituem este documento.

Neste primeiro capítulo, intitulado de "**Introdução**", é inicialmente fornecido ao leitor uma contextualização histórica sobre a vida digital no século *XXI*. De seguida, é disseminada a pequena contextualização fornecida, através da formulação do caso de estudo, onde se estabelece qual o tema deste trabalho. Neste capítulo ainda é possível aferir quais os objetivos que se pretendem alcançar com este trabalho e quais as motivações pessoais e académicas que o sustentam. O capítulo termina com um guia de leitura, que tem por objetivo orientar a leitura deste documento, conforme os interesses de diferentes tipos de leitores.

No segundo capítulo, designado de "**A era Big Data**", é inteiramente focado no tema Big Data. Inicialmente será estudado qual o significado deste conceito e quais as caraterísticas que lhe são inerentes. Depois serão referidos dois componentes tecnológicos de bastante importância no contexto Big Data: MapReduce e Hadoop. Neste capítulo ainda será realizada uma reflexão sobre OLAP num âmbito Big Data e as mudanças que terão de ser levadas a cabo nos armazéns de dados, para que estes possam tirar partido do poder destes dados.

O terceiro capítulo, denominado de "**Bases de dados NoSQL**", será investigado profundamente o mundo NoSQL. Primeiro será fornecida uma introdução a este tema através da explicação do seu significado e de como surgiu o movimento NoSQL. O teorema CAP está na base da filosofia NoSQL, no entanto, para se compreender plenamente esta nova filosofia de armazenamento será necessário fazer uma reflexão sobre as propriedades ACID inerentes aos sistemas relacionais, e o contraste que possuem para as propriedades BASE criadas na sequência do teorema CAP. O foco deste capítulo vai avançando progressivamente até alcançar o estudo sobre os diferentes tipos de bases de dados NoSQL, onde serão levemente abordados alguns sistemas para cada um dos tipos referidos, para além de uma alargada comparação entre as caraterísticas de cada um dos sistemas NoSQL estudados. O capítulo termina com uma breve introdução a um outro novo movimento de armazenamento, não conhecido antes da elaboração desta dissertação, denominado de NewSQL.

O quarto capítulo, que toma o nome de "**Armazéns de dados e NoSQL?**", irá focar-se inicialmente nos critérios de seleção que levaram à escolha dos três sistemas a testar e

comparar nesta dissertação. Para se aferir se é realmente útil substituir as bases de dados relacionais por bases de dados NoSQL, é necessário criar um armazém de dados para servir de comparação. Por conseguinte, a segunda parte deste capítulo apresenta vários estudos que resultaram na criação da estrutura de um armazém de dados relacional, utilizado nesta dissertação. Neste capítulo serão, por fim, também definidos quais os testes de performance a realizar. Antes, todavia, de se chegar a este último tópico será realizada uma ponte entre os temas armazéns de dados e NoSQL através da referência a conceitos inerentes ao desenho de esquemas em bases de dados deste género.

O quinto capítulo, designado de "**Armazéns de dados em MongoDB**", irá focar-se totalmente no sistema *MongoDB*. Nele serão referidos quais os processos e estudos que foram realizados para se conseguir integrar com sucesso um armazém de dados neste sistema. Neste capítulo será primeiramente referida qual a abordagem adotada para analisar a situação proposta. Logo depois desta secção dar-se-á início a uma reflexão sobre as caraterísticas do desenho de um esquema de base de dados no sistema MongoDB. Esta reflexão irá resultar no desenho do esquema do armazém de dados baseado neste sistema, o qual será utilizado nas várias comparações a realizar. Antes ainda de se terminar o capítulo com um estudo sobre o desenho de consultas neste sistema, serão estudados os processos necessários para se migrar e integrar dados de um armazém de dados baseado no sistema MongoDB.

O sexto capítulo, denominado "**Armazéns de dados em Cassandra**", irá incidir na sua totalidade sobre o sistema *Cassandra*. Nele serão referidos quais os processos e estudos que irão ser realizados de modo a conseguir integrar-se um armazém de dados neste sistema. Tal como no capítulo inerente ao sistema MongoDB, serão realizados vários estudos sobre como desenhar corretamente um esquema de base de dados neste sistema. Também serão estudados os vários métodos necessários para se migrar e integrar dados num armazém de dados implementado em Cassandra. Vários problemas e limitações nesta base de dados foram surgindo ao longo da investigação documentada neste capítulo. Assim, a parte final do mesmo será dedicada, não só ao desenho de consultas neste sistema, mas também a ultrapassar todas as dificuldades que surgiram para se conseguir executar as consultas desenhadas no sistema.

O sétimo capítulo, cujo título é "**Testes de Performance",** irá centrar-se nos resultados de performance obtidos com as consultas realizadas em cada sistema. Várias comparações irão ser realizadas para esse efeito recorrendo a gráficos e tabelas ilustrativas dos diferentes tempo de resposta obtidos. Observações e conclusões sobre os resultados em cada sistema serão fornecidas, em simultâneo com algumas explicações sobre fatores que podem ter influenciado ou favorecido os resultados de cada consulta.

No oitavo e último capítulo deste documento, designado de "**Conclusão**" será inicialmente sintetizado toda a investigação realizada neste trabalho de mestrado. Seguidamente será realizada uma reflexão sobre os resultados obtidos com este estudo. Os resultados dos testes de performance realizados no sétimo capítulo, terão uma grande influência sobre as conclusões obtidas, neste capítulo. Para finalizar o capítulo serão elucidadas algumas matérias a estudar como trabalho futuro.

1.6 Guia de Leitura

Este documento deve ser lido na íntegra para se conseguir compreender na totalidade o problema investigado. A estrutura apresentada neste documento segue uma ordem cronológica que tem por objetivo respeitar a estrutura tradicional de uma dissertação de mestrado, ao mesmo tempo espera-se fornecer gradualmente ao leitor a preparação necessária para que possa compreender os assuntos abordados nos capítulos seguintes. No entanto, é esperado que leitores com conhecimentos e objetivos diferentes venham a ler este documento. Por conseguinte, irá ser criado um pequeno guia de leitura, que espera levar diferentes leitores a alcançarem os seus objetivos de uma forma mais rápida e eficiente. Este guia terá por base as palavras-chave definidas no resumo inicial deste documento.

Para o leitor meramente interessando no tema **Big Data,** é recomendado que leia inicialmente o capítulo introdutório, seguido de uma leitura aprofundada do segundo capítulo. Antes de fazer uma leitura do capítulo final deste documento, o leitor deverá ler o Anexo B, onde se encontra um pequeno estudo sobre uma ferramenta destinada a integrar e analisar dados Big Data.

Para o leitor meramente interessado no tema **NoSQL,** a leitura deverá ser diretamente iniciada no terceiro capítulo. É importante que leia a reflexão efetuada sobre o desenho de esquemas em bases de dados NoSQL, presente no quarto capítulo deste documento. Antes de ler o capítulo final, o leitor deverá ponderar a leitura dos capítulos 5 e 6, respeitantes aos sistemas NoSQL, MongoDB e Cassandra.

Para o leitor meramente interessado nos temas **Armazéns de dados** ou **SQL Server 2014**, é recomendado que a leitura se inicie no capítulo introdutório, onde vários assuntos inerentes a armazéns de dados são abordados. Caso não possua conhecimentos sobre o tema Big Data, o leitor deverá ler de seguida o segundo capítulo. Os resultados e ilações tiradas dos testes de performance, presentes no capítulo 7, devem ser analisados antes de se alcançar a conclusão.

Para o leitor meramente interessados no tema **Business Intelligence,** a leitura deve seguir os moldes apresentados no parágrafo anterior, acrescentando a leitura do Anexo B, e ignorando a leitura do capítulo dedicado aos testes de performance.

Para o leitor meramente interessado no tema **MongoDB**, é recomendado que inicie a sua leitura no capítulo 3, onde se reflete sobre algumas caraterísticas deste e outros sistemas NoSQL. O foco do sistema MongoDB neste documento está presente no capítulo 5, pelo que a sua leitura é da maior importância. Antes de terminar a sua leitura no último capítulo, deverá ler o capítulo 7, onde se apresentam os resultados de vários testes de performance realizados sobre este sistema.

Para o leitor meramente interessado no tema **Cassandra**, a leitura deverá ser iniciada no capítulo 3, onde se analisam alguns pontos associados a esta e a outras bases de dados NoSQL. Um estudo complexo realizado sobre o sistema Cassandra, está presente no capítulo 6, pelo

que deve ser objeto de uma leitura profunda. De seguida deverá ler o capítulo 7 onde é apresentado o ambiente de execução e os resultados dos testes operados neste sistema. A leitura deverá terminar no epílogo deste documento.

2 A era *Big Data*

Neste capítulo será investigado o mundo *Big Data*. Será realizada primeiramente uma incursão pela história e significado desta palavra, antes de se investigar alguns conceitos tecnológicos. No final deste capítulo será ainda realizada uma reflexão sobre as mudanças que terão de ser levadas a cabo nos armazéns de dados para que estes possam tirar partido do poder dos dados *Big Data.* Esta dissertação possui na sua essência 3 conceitos chave: *Big Data*, *NoSQL* e Armazéns de Dados. Este capítulo pretende funcionar como o ponto de partida para esta epopeia académica. Para isso fornecendo, não só uma introdução ao conceito Big Data, como também ao tema armazéns de dados. Ao mesmo tempo, este capítulo representa o pano de fundo para a entrada em cena (no próximo capítulo) do conceito *NoSQL*, que tem um peso bastante elevado nesta dissertação, estando neste capítulo contidas ideias e reflexões inerentes aos 3 conceitos chave deste trabalho de mestrado. O estudo que levou à escrita deste capítulo tinha por ambição conseguir responder ao objetivo de fazer uma introspeção profunda sobre o tema *Big Data*.

2.1 O Conceito *Big Data*

Segundo (Ularu et al., 2012) a expressão *Big Data* foi primeiramente utilizada por *Roger Magoulas* quando em 2005, durante uma conferência *Strata*[2], a utilizou para definir aquilo que descrevia como um conjunto de dados que, devido à sua complexidade e tamanho, se encontravam para além do poder apresentado pelas técnicas de gestão de dados tradicionais. Apesar de a expressão ter uma origem relativamente recente, já em 1960 várias organizações, como o *CERN* (Smith, 2013), enfrentavam problemas que hoje são comummente associados ao

[2] Série de conferências, focadas em temas inerentes á ciência dos dados e *Big Data*. São organizadas pelo grupo editorial norte-americano *O'Reilly Media*.

"paradigma" *Big Data*. A diferença, no entanto, está no tamanho dos dados. Na década de sessenta, o problema do tamanho dos dados estava relacionado com dados na ordem dos megabytes, hoje os dados possuem tamanhos que atingem os *terabytes* ou *petabytes* (Borkar et al., 2012).

Mesmo o problema estando presente há já algum tempo, só grandes organizações, como a Google ou Yahoo, possuíam os meios para conseguir extrair informação e conhecimento deste tipo de dados. O grande trabalho requerido para extrair conhecimento destes dados representava um custo megalómano, mesmo para organizações com estas dimensões (O'Reilly Media, 2012). Atualmente, o preço do *hardware* permite que grandes conjuntos de servidores sejam criados com custo relativamente baixo para as organizações. Se associarmos este ponto à consolidação da computação em nuvem e à utilização de *software* de fonte aberta, podemos concluir que até as organizações ou instituições académicas com menos recursos podem experimentar entrar no mundo da gestão e análise de dados *Big Data*.

É importante salientar que o conceito *Big Data* não possui uma definição formal única adotada por todos. De uma forma simples e pragmática, *Big Data* pode ser definido formalmente como um grande volume de dados, que são disponibilizados com diferentes graus de complexidade, gerados a diferentes velocidades e que possuem diferentes graus de ambiguidade; o que resulta numa complexidade que está para além da suportada pelas tecnologias, métodos de processamento e algoritmos "tradicionais" (Krishnan, 2013).

Existem duas grandes fontes de dados que podem ser considerados sob o paradigma Big Data. A primeira, os dados estruturados, semiestruturados e não estruturados que existem no seio das organizações como: correio eletrónico, documentos em formato *PDF*, folhas de cálculo, registos de servidores e outros dados decorrentes da própria atividade da organização. A segunda fonte de dados é o conjunto de dados disponíveis fora das organizações, alguns disponíveis livremente, outros mediante o pagamento de uma subscrição ou disponíveis para grupos restritos de parceiros e/ou clientes selecionados (Sathi, 2012).

Os tipos de dados *Big Data* podem ser divididos em 5 grupos (Soares, 2012):

- **Dados *web* e de redes sociais**

  Dados relativos aos vários tipos de atividades e interações levadas a cabo pelos utilizadores de redes sociais e *blogs*. Nesta categoria também estão incluídos os dados relativos à monitorização e identificação de utilizadores, como dados obtidos através de *cookies* ou *clickstreams*[3].

- **Dados Máquina a Máquina** (*M2M*)

  Dados obtidos através das leituras de vários tipos de sensores ou outros dispositivos capazes de fornecer algum tipo de métrica.

[3] *ClickStream* refere-se ao processo de agregação de informações, sobre que páginas web são visitadas, em que ordem e o tempo que cada utilizador despende em cada uma. Para alcançar esse efeito são seguidas e registadas as sequências e rotas dos cliques que um utilizador realiza numa página *web*.

- **Grandes dados transacionais**

  Dados relativos a registos de telecomunicações (*CDRs*), como chamadas telefónicas, *SMS*, *MMS*. Inclui também dados relativos a registos de faturações, bem como outros dados associados a transações financeiras ou logísticas.

- **Dados biométricos**

  Dados obtidos através da leitura de impressões digitais, scans de retina, deteção de caligrafia, indicadores genéticos e outros tipos semelhantes de dados.

- **Dados Gerados pelo Homem** (*Human-Generated*)

  Esta categoria agrega uma vasta quantidade de dados não estruturados ou semiestruturados, como gravações de voz, mensagens de correio eletrónico, inquéritos, documentos digitalizados e registos ou apontamentos médicos.

2.2 Principais Caraterísticas

Os dados *Big Data* possuem 3 características que, em conjunto, os diferenciam de todos os outros tipos de dados. Essas características são conhecidas como os 3 *Vs* dos dados *Big Data* e são: **v**olume, **v**elocidade e **v**ariedade (Singh e Singh, 2012).

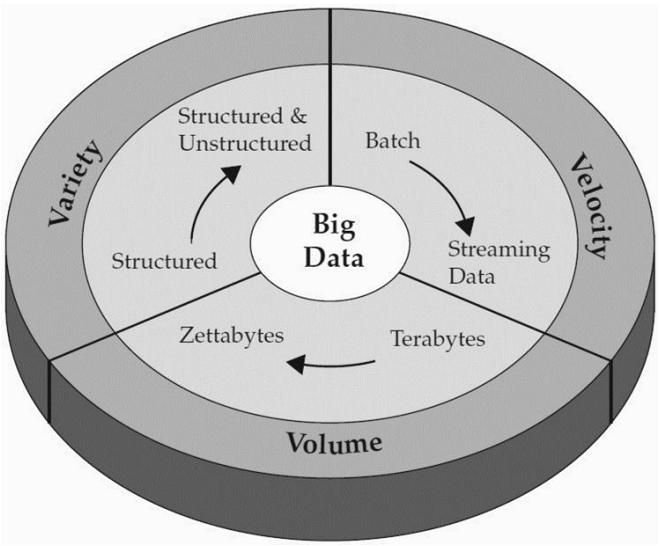

Figura 1 – Características Big Data, os 3 *Vs*. Retirado de (Zikopoulos et al., 2012)

2.2.1 Volume

Estima-se que, desde o início dos tempos até ao ano de 2003, a humanidade tenha armazenado cerca de 5 *exabytes* de dados. Atualmente 2 dias, é o tempo necessário para se armazenar essa

mesma quantidade de dados. No ano de 2012 esse valor tinha alcançado o tamanho de 2.72 *zettabytes* (Sagiroglu e Sinanc, 2013). A previsão é de que antes do ano de 2020 o tamanho dos dados armazenados pela humanidade alcançará os 35 *Zettabytes* (Zikopoulos et al., 2012).

Apesar de todos os valores numéricos e estimativas, a realidade é que a grande parte desses dados não está a ser analisada. A filosofia adotada atualmente passa por armazenar todos os dados com que a organização entra em contacto, na esperança que eles se venham a revelar importantes na descoberta de conhecimento no futuro (Zikopoulos et al., 2012). Uma outra realidade é que, na atualidade, com o advento dos dispositivos inteligentes, quase todos os dispositivos estão a gerar dados, a todas as horas do dia. Mesmo as mais simples atividades diárias da pessoa comum estão a gerar dados; tirar o telemóvel do bolso, mudar de canal, conduzir, tudo exemplos de ações que geram dados. Num futuro bastante próximo, as nossas casas, os nossos eletrodomésticos e devido ao professado sucesso das tecnologias utilizadas como acessórios de vestuário, até os nossos corpos irão gerar dados.

O facto de que todos estes dados precisam de estar acessíveis ao mesmo tempo que se apresentam pesquisáveis, processáveis e gerenciáveis tornam o **V**olume na característica mais importante e desafiadora no mundo *Big Data* (Demchenko et al., 2013).

2.2.2 Velocidade

É bastante difícil e desafiador conseguir gerir todos os dados que chegam a grande velocidade aos repositórios das organizações. Para se tirar partido de todas as potencialidades dos dados *Big Data*, os dados necessitam muitas vezes de ser analisados "ainda em movimento e não quando já se encontram a descansar" (Zikopoulos et al., 2012). O facto de os dados terem de ser analisados em breves instantes, acaba por se tornar num problema para algumas organizações, pois a velocidade com que os dados chegam, ultrapassa em muito as capacidades disponíveis para o processamento desses dados.

Muitas organizações possuem necessidade de, em milissegundos, conseguirem identificar se os dados são importantes, se merecem ser profundamente analisados, ou se esses dados precisam de ser conjugados com outros dados, para assim fornecerem um conhecimento mais relevante (Gerhardt et al., 2012). Se uma câmara de vigilância localizar um criminoso procurado, um evento tem de ser gerado pouco tempo depois de as imagens terem sido obtidas. As organizações que suportam redes sociais têm a necessidade de gerar publicidade personalizada, segundos depois de um "gosto" ter sido colocado na fotografia de um produto específico. Anomalias têm que ser detetadas nos dados antes de estes serem armazenados, para que ações em tempo real possam ser tomadas (Gerhardt et al., 2012).

2.2.3 Variedade

Os dados *Big Data* não se apresentam apenas sob uma forma tradicional (estruturada). Devido à grande utilização de sensores, dispositivos inteligentes, redes sociais e *blogs*, a grande parte

destes dados apresenta-se sob uma forma semiestruturada ou completamente não estruturada. Estes dados são "aleatórios, enormes, e difíceis de analisar" (Gerhardt et al., 2012).

Esta variedade coloca alguns problemas e desafios aos sistemas tradicionais, que não são adequados para executar as requeridas análises e extrair conhecimento de dados com tamanha complexidade. Os dados sob um formato "relacional" e estruturado, representam apenas 20% dos dados disponíveis no mundo (Zikopoulos et al., 2012). As possibilidades de extrair conhecimento, de dados semi ou não estruturado, são imensas para as organizações "corajosas" o suficiente para embarcar neste novo paradigma. Embarcar plenamente neste paradigma implica uma mudança tecnológica e uma mudança de filosofias empresariais que as organizações têm de levar a cabo. Num mundo onde uma décima de vantagem para com a concorrência é um indicador de sucesso, 80% representa um valor demasiado elevado de dados por analisar e, por consequência, de conhecimento que não pode ser deixado insondado.

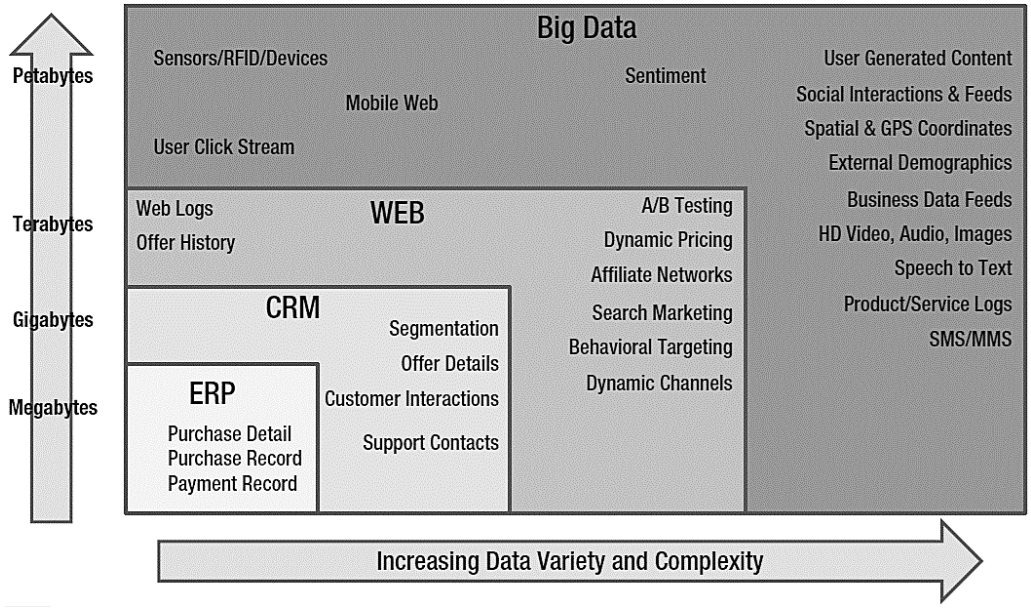

Figura 2 – Fonte de dados *versus* volume e variedade. Retirado de (Mohanty et al., 2013)

2.2.4 Valor e Veracidade

Existem autores (Demchenko et al., 2013) (Sathi, 2012) que advogam a adição de mais *Vs* aos 3 normalmente considerados como característicos dos dados Big Data. Estes autores consideram que o **V**alor e a **V**eracidade dos dados também são duas características que distinguem os dados *Big Data* de todos os outros tipos de dados.

A maioria dos dados *Big Data* têm proveniência fora do controlo da organização. Desse modo a **V**eracidade dos dados torna-se bastante relevante, pois decisões importantes irão ser tomadas baseadas no conhecimento obtido com esses dados. A veracidade no contexto *Big Data* representa a confiabilidade dos dados, a credibilidade da fonte desses dados e também a adequação dos mesmos ao público-alvo pretendido (Sathi, 2012). A veracidade em *Big Data*

assegura que os dados são "confiáveis, autênticos e protegidos de acessos e modificações não autorizadas" (Demchenko et al., 2013) . Para assegurar a veracidade desses dados, deve ser tomada especial atenção aos processos que asseguraram a segurança dos dados ao longo do seu ciclo de vida. Os dados devem possuir fontes confiáveis, e devem ser processados e armazenados em estruturas confiáveis (Demchenko et al., 2013).

O **V**alor é uma característica muito importante dos dados. Este é definido pelo conhecimento e informação que os dados trazem para o processo de análise. O valor a retirar de um grupo de dados irá definir qual a quantidade de dados que deve ser retida, bem como o período de tempo que esses dados devem ser armazenados. É possível que um grupo de dados só se revele importante quando conjugado com outros dados, relativos a eventos que ainda não aconteceram, ou apenas quando um padrão semelhante emergir. Desse modo, o valor dos dados está "intimamente relacionado com o volume e variedade dos mesmos" (Demchenko et al., 2013).

2.3 Conceitos Tecnológicos

2.3.1 MapReduce

MapReduce é um modelo de programação desenvolvido pela Google e apresentado no artigo (Dean e Ghemawat, 2004). Nesse artigo não estava definido só o modelo de programação, mas também parte da implementação específica utilizada pela Google, recorrendo a esta *framework*. Este modelo tinha como objetivo simplificar o desenvolvimento e implementação de aplicações de processamento de dados de larga escala, distribuídas e tolerantes a falhas (Sammer, 2012).

As primitivas para se alcançar o processamento de dados com *MapReduce* são denominadas por *mapers* e *reducers* (Lam, 2011). Assim, aplicações desenvolvidas segundo este modelo requerem que os programadores criem duas funções, *map* e *reduce*. O sistema fica depois encarregue de gerir a paralelização, o agendamento e distribuição da carga de trabalho por cada nó de processamento, bem como a monotorização de todo processo e a respetiva recuperação e tratamento de falhas, se estas ocorrerem (Sammer, 2012).

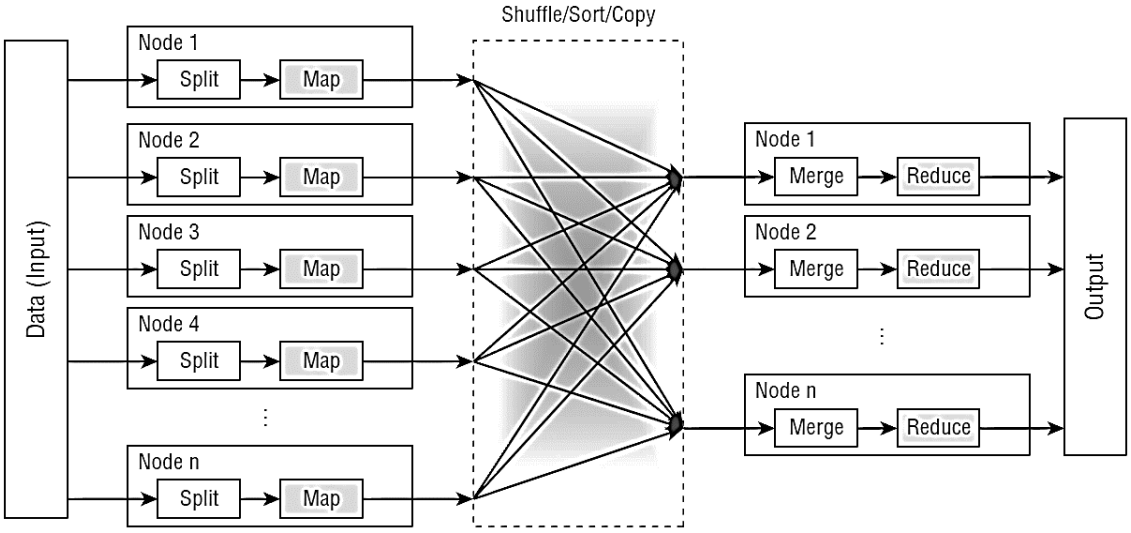

Figura 3 – Fluxo de um processo MapReduce. Retirado de (Schmarzo, 2013)

Como pode ser observado na figura anterior, este modelo baseia-se em 5 operações chave (Loshin, 2013):

- **Chegada dos dados** (*Data input*)**:** Os dados são carregados para o sistema e são seguidamente distribuídos pelos nós de armazenamento disponíveis.

- **Mapeamento** (*Map*)**:** Os dados em cada nó são divididos em blocos. Cada bloco irá ser atribuído a um nó de processamento que irá correr uma função *map* sobre esses dados. As funções *map* são desenhadas para executar uma tarefa específica e retornar uma lista de pares chave-valor. Por exemplo, uma função map encarregue de contar o número de ocorrência de cada palavra que existe num grupo de dados devolveria uma lista parecida com a seguinte: (assim, 16); (então, 6); (foi, 11) … O par chave-valor tem como chave a palavra contada e como valor o número de ocorrências da mesma.

- **Organização** (*Shuffle*/ *Sort*)**:** Existem vários nós de processamento a executar funções *map* e, por consequência, irão existir várias listas contendo pares chave-valor. A função executada neste ponto tem por objetivo garantir que todas as contagens de ocorrências de uma palavra a nível global estão na mesma lista. Por exemplo, a função de *shuffle* recebe 3 listas, vindas de 3 nós de processamento diferentes: **1-** (assim, 16); (então, 6); (foi, 11) **|| 2-** (assim, 10); (foi, 5) **|| 3-** (assim, 9); (então, 12); (foi, 17); e terá como resultado as seguintes 3 listas:

  (assim, 16); (assim, 10); (assim, 9);
  (então, 6); (então, 12);
  (foi, 11); (foi, 17); (foi, 5)

- **Reduzir** (*Reduce*)**:** Vão existir vários nós a executarem funções *reduce*, e cada um irá receber uma lista e contar as ocorrências de cada palavra nessa lista. Por exemplo, uma função *reduce* recebe a seguinte lista: (assim, 16); (assim, 10); (assim, 9) o resultado

seria (assim, 35). De notar que, neste exemplo, uma lista possui apenas ocorrências de uma palavra, mas uma mesma lista pode possuir ocorrências de várias palavras, desde que todas as ocorrências de cada palavra se encontrem presentes nessa mesma lista.

- **Resultado final** (*output*)**:** Todos os resultados das várias funções *reduce* irão ser conjugados e ordenados num único resultado final. Um exemplo ilustrado do problema aqui apresentado pode ser observado na imagem seguinte.

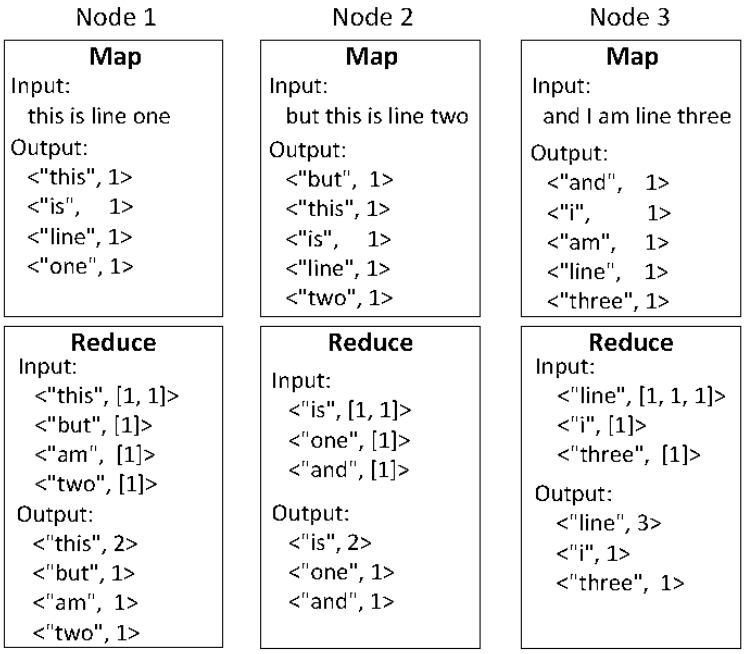

Figura 4 – Entrada e saída de dados de tarefas MapReduce. Retirado de (Silva et al., 2014)

A programação focada neste modelo não requer a escrita de código complexo e fastidioso, podendo o programador dedicar-se à lógica do negócio. No entanto, criar aplicações recorrendo ao modelo *MapReduce* pode revelar-se um processo muito pouco trivial. A vantagem está no facto de que, após as aplicações estarem desenvolvidas, fazer com que a aplicação seja executada por milhares ou mesmo milhões de máquinas, requer apenas uma pequena mudança de configurações (Lam, 2011). O *Hadoop MapReduce* é uma implementação específica e de código aberto deste modelo, e será abordada de forma resumida na próxima subsecção desta dissertação.

2.3.2 Hadoop

Após a consulta e estudo das fontes e conteúdos adquiridos em relação ao tema *Big Data*, foi facilmente salientável que a tecnologia *Hadoop* era referida por uma grande parte dos autores. O estudo mais aprofundado levou à conclusão de que é impossível tocar o tema *Big Data*, neste momento, e não referir *Hadoop*. Esta é, de momento, a tecnologia de referência adotada no

âmbito *Big Data*. *Hadoop* é um projeto da *Apache Software Foundation*[4], escrito em Java. Inicialmente desenvolvido como um complemento para o sistema *Nutch*[5], veio com o tempo a ganhar destaque e tornou-se um projeto de alto nível dentro do grupo de projetos *Apache*. A importância e relevância deste sistema pode, em grande parte, ser avaliada pelo número de subprojectos que ao longo do tempo veio a albergar em si. De uma forma pragmática, *Hadoop* pode ser definido como uma *framework* de fonte aberta destinada à criação e execução de aplicações distribuídas, com a intenção de processar grandes quantidades de dados (Lam, 2011). O *Hadoop* apresenta 4 elementos chave que o distinguem de outras soluções (Lam, 2011):

- **Acessível:** O *Hadoop* tanto pode ser executado numa grande quantidade de servidores compostos por máquinas de preços acessíveis, como através da computação em nuvem.
- **Robusto:** Como foi desenhado para ser executado recorrendo a *hardware* relativamente barato, a arquitetura do *hadoop* está preparada para gerir a grande maioria das avarias de *hardware* que possam surgir.
- **Escalável:** O *Haddop* é linearmente escalável, bastando apenas adicionar mais nós de processamento ao conjunto de recursos, para assim se conseguir processar uma maior quantidade de dados.
- **Simples:** É bastante rápido escrever código eficiente e de processamento paralelo.

Este projeto possui 2 componentes principais (Patel et al., 2012). O primeiro é o *Hadoop Distributed File System* (*HDFS*). Como o próprio nome indica, este constitui um sistema de ficheiros, que serve como componente de armazenamento e que representa a base deste modelo (Holmes, 2012); foi moldado à semelhança do sistema de ficheiros criado pela *Google*, o *Google File System* (*GFS*) apresentado em (Ghemawat et al., 2003).

O *HDFS* foi desenhado para guardar ficheiros de tamanhos elevados, podendo albergar *terabytes* de dados por ficheiro. Este sistema de ficheiros foi pensado em torno da ideia de que o melhor padrão de *software* para processamento paralelo de dados é: escrever uma vez, ler muitas vezes. Isto significa que os conjuntos de dados são copiados da fonte para o sistema, e depois são então analisados várias vezes ao longo do tempo. Cada análise irá requerer uma grande proporção de um conjunto de dados, se não mesmo o conjunto inteiro; assim é mais importante o tempo que o sistema demora a ler o conjunto de dados inteiro, que a latência necessária para ler o primeiro registo desses dados (White, 2012). Desse modo, e considerando que este sistema é otimizado para alcançar grandes velocidades de transferência, a sua utilização não é recomendada para aplicações que requeiram pouca latência no acesso aos dados. O outro componente principal deste sistema é o *Hadoop MapReduce*, criado à imagem do modelo apresentado pela *Google* e resumidamente já explicado na secção 2.3.1 desta dissertação.

[4] A *Apache Software Foundation* é uma organização sem fins lucrativos norte-americana, fundada com o objetivo de suportar os projetos Apache.
[5] O *Apache Nutch* é uma um motor de pesquisa web de código aberto, baseado no projeto *Apache Lucene* e desenvolvido na linguagem *Java*.

Segundo (Cuzzocrea et al., 2011) o ecossistema *Hadoop* enquadra-se nos 3 pontos *MAD* apresentadas por (Cohen et al., 2009) para descrever as características que definem um sistema capaz de operar e tirar dividendos de dados *Big Data*. Para a fonte citada, *Hadoop* é *MAD* porque:

- **M**agnetismo: é capaz de atrair e trabalhar com todas as fontes de dados.
- **A**gilidade: é capaz de adaptar os seus componentes principais às evoluções que possam ocorrer nas fontes de dados *Big Data*.
- Profundidade (***D**epht*): é capaz de suportar análises complexas sobre fontes de dados *Big Data*, muito para além das capacidades apresentadas pelas tradicionais ferramentas de análise de dados baseadas em *SQL*.

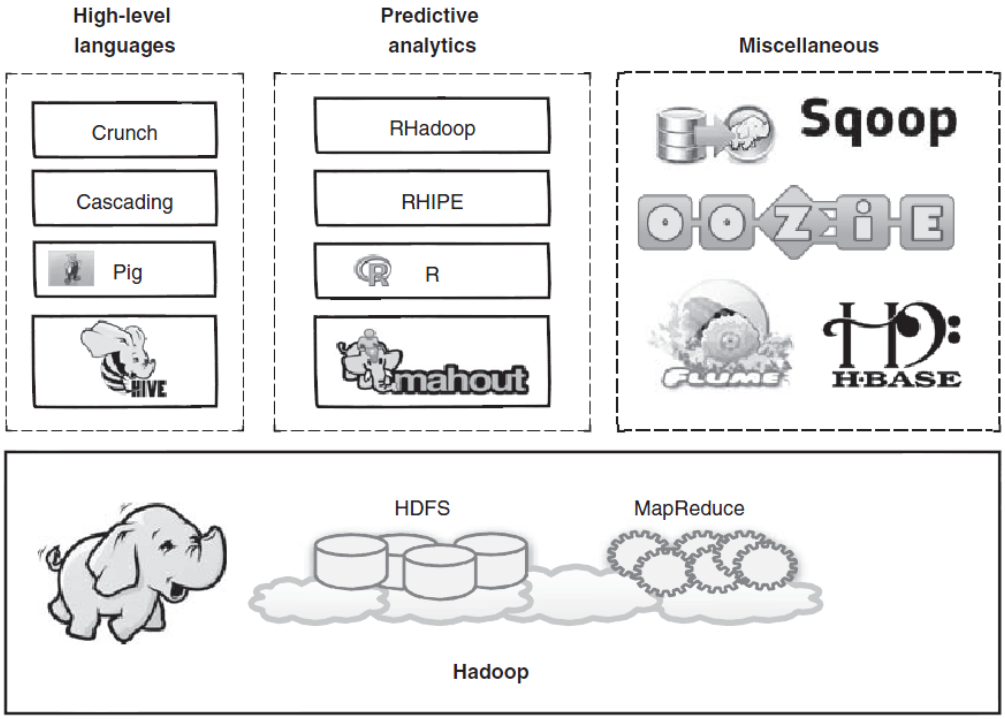

Figura 5 – O ecossistema *Hadoop*. Retirado de (Holmes, 2012)

Actualmente o *Hadoop* possui um ecossistema bastante complexo, com bastantes soluções capazes de estender as funcionalidades do núcleo *Hadoop* em diferentes contextos. No entanto, todas essas aplicações continuam a ter por base o *HDFS* e/ou o *Hadoop MapReduce*, visto que estas representam o núcleo duro desta tecnologia. Nesta dissertação será abordado com maior profundidade na secção 6.4.1, o projeto *Apache Hive*.

2.4 Business Intelligence na era *Big Data*

2.4.1 Contextualização

O mundo organizacional de hoje está construído à volta dos dados e do poder que estes fornecem. A vida de uma organização é altamente influenciada pela sua capacidade de gerir e extrair informação e conhecimento relevante dos dados que possui (Ularu et al., 2012). Os resultados de todos os processos de *Business Intelligence* são uma ajuda essencial no processo de tomada de decisão. A cada vez maior presença de dados *Big Data* no dia-a-dia das organizações significa que existe um vasto conhecimento que está ao alcance mas ao mesmo tempo inexplorado. Apesar dos desejos das organizações, os processos e tecnologias *Business Intelligence* ainda não se encontram preparados para conseguir domar este novo paradigma. A integração de dados *Big Data* e dados operacionais num mesmo repositório, poderá conduzir a análises e conhecimentos inimagináveis até ao momento.

Para tornar os dados *Big Data* numa vantagem para o negócio, as organizações terão de repensar a maneira como gerem os dados dentro dos seus repositórios de dados. Tradicionalmente, os processos de gestão e análises de dados seguem uma estrutura bem definida e sequencial que tem vindo a ser aprimorada e melhorada ao longo de vários anos. Primeiro definem-se os requisitos do negócio, de seguida desenvolvem-se os módulos de integração de dados, para assim se incluírem dados de vários sistemas empresariais (Vendas, *Marketing*, Finanças, *CRM*, *ERP*); o passo seguinte implica analisar e gerar perfis dos dados, para verificar se estes se encontram completos e corretos. Depois de os dados estarem agregados e validados é necessário criar modelos orientados à execução de análises (armazéns de dados empresariais, *data marts*); sobre eles são realizadas análises condizentes com os objetivos do negócio; estas análises resultam em relatórios, gráficos e cubos multidimensionais que podem depois ser explorados de diferentes modos, para se obterem diferentes informações. A grande influência da *web* e dos dispositivos móveis está a colocar alguns desafios a este modelo e processos tradicionais. O grande fator por detrás desta ocorrência é a mudança na natureza dos dados. Os dados são cada vez menos centralizados e deixam de estar meramente limitados aos sistemas das organizações.

2.4.2 Armazéns de dados na era Big Data

O conceito de armazém de dados foi introduzido no início da década de 90 do século XX, por (Inmon, 1992). Desde então, têm surgido várias tecnologias, paradigmas e implementações esmiuçadas por investigadores académicos e empresariais. Simultaneamente, a taxa de adoção destes sistemas tem-se tornado cada vez mais elevada. A realidade é que o conceito armazém de dados possui uma definição ampla dependendo do contexto utilizado. Para (Chaudhuri e Dayal, 1997), um armazém de dados é um conjunto de tecnologias e métodos de apoio à decisão, vocacionados para apoiar os "profissionais do conhecimento" a tomarem melhores e

mais rápidas decisões no seio das organizações que suportam. Já (Chaudhuri e Dayal, 1997), na sua definição apresenta um foco no aspeto armazenamento. Para a fonte citada um armazém de dados é uma coleção de dados, não voláteis, variáveis no tempo, orientados para um tópico específico e é principalmente utilizado na tomada de decisões organizacionais.

Durante as últimas décadas, os sistemas de armazéns de dados empresariais têm sido considerados como o braço direito das organizações no que diz respeito à extração de informação e conhecimento. O movimento *Big Data* representa uma grande oportunidade para todas as organizações. No entanto, adotar este paradigma implica a utilização de estilos de implementação e tecnologia diferente daquela utilizada tradicionalmente pelos armazéns de dados empresariais. Para conseguir tirar partido dos dados *Big Data* e deste movimento que se prevê a próxima fonte de riqueza informacional, as organizações terão de desenvolver e implementar um ecossistema/plataforma que utilize os armazéns de dados tradicionais conjugados com dados *Big Data*. Para alcançar esse efeito terão de recorrer a soluções híbridas, meticulosamente desenvolvidas para o efeito. Seguidamente, será apresentada uma comparação entre alguns conceitos inerentes a armazéns de dados tradicionais e armazéns de dados capazes de albergar dados *Big Data* (Mohanty et al., 2013):

- **Expetativas empresariais**
    - ***Armazém de dados Big Data***

      Principalmente focados em análises exploratórias que possibilitem encontrar novas perspetivas e um rápido e pronto acesso a novos dados. A veracidade dos resultados pode, ou não, ser questionável; uma vez que o valor e veracidade dos dados não são apurados e a qualidade dos mesmos não é controlada antes destes chegarem à organização.

    - ***Armazém de dados Empresarial***

      Todos os dados são estritamente verificados e é garantida a sua qualidade e veracidade. Maioritariamente baseados em factos, são geridos e organizados de forma a responderem a requerimentos específicos de análises orientadas ao modelo de negócio da organização. Em muitos casos, o armazém de dados empresarial é considerado como a única fonte de veracidade numa empresa.

- **Metodologia de Desenho**
    - ***Armazém de dados Big Data***

      Abordagem altamente ágil e interativa que possibilita a criação rápida de conhecimento. Conseguido através da agregação do maior número de fontes de dados possível, ao mesmo tempo que se executam o maior número possível de algoritmos sofisticados. O objetivo é provar ou validar hipóteses de negócio, para isso combinando dados de diferentes fontes, originados dentro e fora da organização, com ou sem, estrutura e modelo de dados bem definidos.

    - ***Armazém de dados Empresarial***

      Utiliza uma combinação de metodologias definidas após as necessidades de dados terem sido avaliadas cuidadosamente, tendo em conta aspetos relacionados com a

integração e utilização desses dados. São desenhadas de modo a serem aplicadas aos requerimentos específicos do negócio.

- **Considerações de Arquitetura**
  - **Armazém de dados Big Data**

    Deve ser capaz de conseguir integrar todo o tipo de dados, independentemente da estrutura que apresentem. Deve possuir a capacidade de ser facilmente escalável e a um baixo preço. Deve ter a capacidade de analisar grandes volumes de dados, sem ter de recorrer a mecanismos de amostragens.

  - ***Armazém de dados Empresarial***

    Nem todos os dados são administrados e mantidos dentro do armazém de dados. As fontes de dados são previamente conhecidas, e a qualidade dos dados é previamente assegurada, de modo a satisfazer requisitos específicos do negócio da organização. Tem a possibilidade de ser escalável, mas com um grande custo acrescentado. Arquiteturas MPP[6] são utilizadas para se conseguir alcançar um elevado desempenho. Os dados são periodicamente arquivados, de modo a acomodar o crescimento de dados e para manter o custo de armazenamento baixo.

- **Standards e Integridade dos dados**
  - **Armazém de dados Big Data**

    Standards de integração de dados são vagamente definidos, maioritariamente orientados pelo programador ou pelo estilo da aplicação. Não possuem métodos de gestão de meta dados e regras de negócio.

  - ***Armazém de dados Empresarial***

    Regem-se pelos princípios dos sistemas de gestão de bases de dados relacionais, e as respetivas abordagens no que diz respeito à arquitetura e consistência dos dados (integridade relacional, regras de negócio). Os dados são normalmente centralizados, e os programas de processamento de dados seguem uma abordagem de execução bem definida, na maioria dos casos a execução de programas concorrentes é realizada de forma sequencial.

2.4.3 Análises de dados na era Big Data

As análises de dados fornecem uma plataforma e uma oportunidade para medir tudo o que acontece numa organização. Os vários processos e métodos de análise têm por objetivo conseguir fornecer aos elementos decisores de uma organização as melhores perspetivas sobre potenciais oportunidades, ameaças, riscos e problemas que possam surgir. Este conhecimento e compreensão é algo essencial no processo de tomada de decisão no seio de uma organização.

[6] ***M**assively **P**arallel **P**rocessing* (processamento paralelo massivo) refere-se à utilização de um elevado número de processadores, para se executar um conjunto coordenado de computações, de forma paralela.

A análise de dados *Big Data* pode ser definida como uma combinação de técnicas de análise e descoberta de conhecimento tradicionais, conjugadas com grandes volumes de dados. O seu objetivo passa por criar uma plataforma fundacional para analisar, modelar e prever o comportamento de clientes, mercados, produtos, serviços e competidores. Estes fatores permitem à organização definir estratégias à medida das necessidades e ambições da empresa, para o sector de mercado e clientes que pretende alcançar (Krishnan, 2013).

Quando conjugados na mesma linha de raciocínio é impossível desassociar os temas armazéns de dados, análises de dados e *OLAP*. O Processamento analítico de dados (*OLAP*) representa um conjunto de técnicas de exploração de bases de dados que permitem ao utilizador obter de forma eficiente uma agregação de dados específica (Chen e Ordonez, 2008). A agregação dos dados é utilizada para gerar maior ou menor nível de detalhe. Por baixo da estrutura visual das aplicações *OLAP* está presente uma malha multidimensional, que armazena agregações para todos os níveis de detalhe (Chen e Ordonez, 2008). De um ângulo de "visão" diferente, essa malha é considerada como um cubo multidimensional. Um cubo *OLAP* permite a análise centrada em agregações de dados transacionais, ao moldar os registos de dados em factos mensuráveis com caraterísticas dimensionais (Mansmann et al., 2014). Uma vista multidimensional é obtida através dos campos de dados disponíveis e das relações explícitas que existem entre eles. Este modelo clássico não é viável quando aplicado a cenários em que é necessário trabalhar com dados pouco estruturados (Mansmann et al., 2014).

O conceito *OLAP*, introduzido já há mais de duas décadas por (Codd et al., 1993), e os armazéns de dados têm vindo a evoluir conjuntamente e atingiram maturidade tecnológica ultrapassando para isso várias dificuldades e limitações (Abell et al., 2011). No entanto, alguns desafios ainda se encontram no horizonte, entre eles, a cada vez mais emergente necessidade de conseguir gerir e analisar dados *Big Data*. Em (Cuzzocrea et al., 2011) e (Cuzzocrea et al., 2013) são elucidados alguns dos problemas, desafios e questões relacionadas com análises de dados gerais e *OLAP* sobre dados *Big Data,* dos quais se salientam os seguintes:

- **Tamanho:** Quando alojam dados *Big Data* as tabelas de factos podem facilmente alcançar tamanhos bastante elevados. Esta situação traz grandes problemas computacionais, e o tamanho destas tabelas pode tornar-se num fator que coloca em causa a performance de aplicações que efetuam operações sobre essas tabelas.

- **Complexidade:** A construção de cubos *OLAP* sobre dados *Big Data*, acarreta um grau elevado de complexidade, que não está presente numa situação homóloga com dados relacionais. Devido à natureza não estruturada dos dados *Big Data*, o número de tabelas dimensionais num esquema pode alcançar valores bastante elevados. Ao mesmo tempo, nestes cubos pode existir a necessidade de representar medidas múltiplas e heterogéneas.

- **Incongruência e heterogeneidade das fontes de dados:** os dados *Big Data* têm na sua origem várias fontes de dados; esses dados variam em grande parte no conteúdo e na forma como são apresentados. Esta situação traz alguns problemas de integração, pois é necessário moldar e limpar os dados de modo a que eles assentem no esquema do armazém de dados. Devido ao grande tamanho dos repositórios *Big Data,* é essencial filtrar os dados não essenciais e que não estão enquadrados no âmbito das análises a realizar. Este processo revela-se um ponto crítico para a qualidade dos resultados finais obtidos com as análises *OLAP* efetuadas.

- **Desempenho para o utilizador final:** Devido ao tamanho e complexidade dos cubos *OLAP* de dados *Big Data*, a performance computacional pode revelar-se um fator crítico para os utilizadores que realizam análises, especialmente durante as fases de agregação e consulta dos dados. Desde modo, torna-se especialmente relevante ter em consideração aspetos como a usabilidade de tais cubos. Revela-se extremamente importante definir métodos que permitam avaliar se é viável analisar um determinado cubo, ao mesmo tempo que também é necessário definir métodos capazes de limitar a complexidade destes cubos ao mínimo.

2.5 Conclusões

O conceito *Big Data* revelou-se bastante mais vasto e evasivo do que se pensava previamente. O mundo em que se insere está num constante e irrequieto movimento, de tal modo que novos desenvolvimentos devem ser esperados semanalmente. É igualmente esperado que o conceito e as tecnologias nele incluídas sofram bastantes mudanças evolutivas nos próximos anos. Acaba por ser um sentimento agridoce documentar este tema, visto que este estará constantemente em "movimento". Assim, focaram-se os aspetos mais imutáveis e importantes deste conceito, esperando contribuir com uma investigação que consiga resistir no tempo à efervescência do movimento *Big Data*.

A investigação final sobre BI na era Big Data permitiu retirar várias ilações que não eram esperadas, sendo a mais significativa o facto de que a simples utilização de bases de dados *NoSQL,* em armazéns de dados, não será suficiente para controlar o paradigma *Big Data*. Na realidade, para se conseguir retirar conhecimento efetivo destes dados, várias mudanças têm de ser operadas, e não só a um nível tecnológico. A questão *NoSQL* representa apenas um dos vários pontos que necessitam de ser analisados no contexto *Big Data* em Armazéns de Dados. Este estudo acabou por levantar algumas dúvidas e curiosidades. Qual é o estado de desenvolvimento e como operam as ferramentas destinadas a integrar e analisar dados *Big Data*? É inequivocamente interessante conseguir realizar consultas *OLAP* visuais sobre estes dados, será que é mesmo possível? Com vista a responder a estas questões foi realizado um pequeno estudo, com foco numa ferramenta específica, o qual pode ser encontrado no Anexo B deste documento.

3 Bases de dados *NoSQ*L

Este capítulo pretende constituir uma investigação profunda sobre o conceito *NoSQL*, e as tecnologias a ele associadas. O estudo deste novo paradigma de armazenamento representa um dos pontos mais importantes deste projeto investigativo e, por conseguinte, este ponto foi alvo de um estudo bastante profundo e demorado. Espera-se com este capítulo, conseguir estabelecer um plano de fundo, sobre o qual serão trabalhadas as implementações locais dos diferentes sistemas NoSQL a seleccionar.

3.1 Movimento *NoSQL*

Durante décadas, as bases de dados relacionais foram utilizadas para armazenar dados estruturados. Os dados estão divididos em vários grupos que tomam o nome de tabelas. Estas tabelas possuem várias unidades bem definidas de dados em termos de tipo, tamanho e outras restrições. Cada unidade de dados representa uma coluna e cada unidade do grupo representa uma linha (Vaish, 2013). As bases de dados relacionais devem o seu nome às várias relações que existem entre colunas e tabelas. Com o aparecimento das grandes empresas *web*, surgiu a exigência de se processar grandes quantidades de dados, cada vez mais não estruturados, de uma forma eficiente e barata. Assim, surgiu a necessidade de se escalar dados horizontalmente. Isso significa aumentar o número de unidades de armazenamento, numa tentativa de aumentar o poder de processamento (Bernardino e Abramova, 2013). Como a integridade e consistência são fatores críticos de sucesso, as bases de dados relacionais não apresentam os requisitos de escalabilidade necessários para suportar grandes volumes de dados transacionais (Krishnan, 2013). Surgiu então o movimento *NoSQL*, cuja ascensão está intimamente ligada ao crescimento das grandes empresas *web* (Moniruzzaman e Hossain, 2013). Sendo que grande parte dos projetos principais no âmbito *NoSQL*, têm a sua base em projetos criados pela *Google* (*BigTable*), *Facebook* (*Cassandra*) e *Amazon* (*Dynamo*).

Se traduzido literalmente *NoSQL* representa duas palavras: *No* (não) e *SQL* (*Structured Query Language*). O nome devia-se à filosofia dos seus criadores, que pretendiam criar um modelo distinto do modelo relacional e dos seus princípios *ACID*. Com o passar do tempo o nome acabou por se tornar numa representação errada do conceito. Atualmente o nome é interpretado como "*Not Only SQL*" (não apenas *SQL*). O conceito *NoSQL* não se refere a um produto ou tecnologia específica, mas sim a um conjunto de termos diversos, por vezes relacionados, que se encontram associados ao rápido e eficiente processamento de dados, com foco principal na performance, confiança, segurança e agilidade (McCreary e Kelly, 2014).

Seguidamente serão enunciadas algumas caraterísticas e informações, representativas do paradigma *NoSQL*:

- **Representação de dados sem esquema**
  Quase todas as implementações *NoSQL* apresentam representações de dados sem esquema, isso significa que não é necessário pensar muito à frente e imaginar as necessidades de dados futuras para que seja plausível definir e criar a estrutura da base de dados. É sempre possível ir evoluindo ao longo do tempo, e adicionar novos atributos à medida que a necessidade se verifique (Vaish, 2013). Esta abstração permite que seja apenas necessário colocar um ficheiro numa pasta do sistema, para se poderem executar consultas a esses dados (McCreary e Kelly, 2014). Sistemas *NoSQL* são capazes de armazenar dados em vários formatos diferentes: grafo, par chave/valor, Documento e colunas. Estes diferentes tipos de bases de dados *NoSQL* serão abordados em profundidade na secção 3.4 desta dissertação.

- **Tempo de desenvolvimento**
  O tempo de desenvolvimento necessário é bastante mais curto, devido ao facto de existirem *APIs* de fácil utilização, que não requererem ao utilizador a criação de comandos *SQL* complexos e intrínsecos (Vaish, 2013). A grande maioria dos sistemas *NoSQL* permitem extrair informações dos dados, recorrendo a interfaces simples sem ser necessário recorrer à utilização de *JOINs* ou outras funções *SQL* (McCreary e Kelly, 2014).

- **Velocidade e Escalabilidade**
  Os sistemas *NoSQL* permitem que os dados sejam trabalhados por múltiplos processadores, mantendo a mesma performance de alta velocidade. A maioria destes sistemas utiliza o paradigma "nada partilhado", assim são utilizados vários processadores, de preços acessíveis, que recorrem a memória RAM e disco rígido não compartilhados. Os sistemas *NoSQL* possuem uma escalabilidade linear, quando se adicionam mais processadores, obtém-se um aumento consistente de performance (McCreary e Kelly, 2014).

3.2 Propriedades ACID

O controlo de transações é um aspeto bastante importante a considerar, quando se pensa na performance e consistência de bases de dados implementadas num ambiente de computação distribuída (McCreary e Kelly, 2014). Durante bastante tempo, os sistemas de gestão de base de dados relacionais que garantiam as propriedades *ACID* (**A**tomicidade, **C**onsistência, **I**solamento e **D**urabilidade) dominavam o mercado e supriam a grande maioria das necessidades de armazenamento apresentadas pelas organizações (Simon S.Y. Shim, 2012). Todavia, a nova era dos dados gerados pela web e a consequente explosão na quantidade e tamanho dos dados armazenados pelas organizações (*Big Data*), levou a que diferentes necessidades de armazenamento surgissem (*NoSQL*), bem como uma mudança no paradigma de controlo de transações (*BASE*). Para se compreender as motivações que levaram a esta mudança de paradigma, é necessário compreender as implicações do modelo de controlo de transações *ACID*:

- **Atomicidade**
  Ou uma transação operacional ocorre com sucesso ou falha. Não existe um estado intermédio que seja aceitável (Tiwari, 2011). Se uma parte da transação falha, toda a transação falha; só se todos os componentes da transação forem bem-sucedidos é que a transação é considerada bem-sucedida. Se algum componente da transação falhar, todas as alterações efetuadas até àquele ponto têm de ser revertidas e a base de dados tem de voltar ao estado em que se encontrava antes de se iniciar a transação. A atomicidade tem que ser assegurada em toda e qualquer situação, como problemas de *hardware/software* ou falhas de energia (Mohanty et al., 2013).

- **Consistência**
  A consistência assegura que uma transação efetuada irá deixar a base de dados num estado válido. Isso significa que todos os dados escritos na base de dados necessitam de ser validados segundo as regras definidas (Leonard, 2013). Assim, os dados inseridos têm de respeitar restrições impostas pelo esquema da base de dados, tipos de dados e integridade referencial de tabelas ou linhas (Mohanty et al., 2013).

- **Isolamento**
  O fator isolamento torna-se relevante quando os mesmos dados são acedidos por dois processos concorrentes (Tiwari, 2011). Cada transação tem de ser executada em total isolamento. Se duas transações estão a ocorrer ao mesmo tempo, elas devem permanecer sem conhecimento uma da outra (Mohanty et al., 2013). Dependendo do nível de isolamento imposto, isso pode significar que o mecanismo de gestão da base de dados pode necessitar de bloquear temporariamente a execução de uma transação até que outra tenha terminado (Dimiduk e Khurana, 2013).

- **Durabilidade**
  A durabilidade assegura que, após uma transação ser executada com sucesso, o seu resultado é guardado permanentemente (Tiwari, 2011), mesmo na eventualidade de erros ou falhas de sistema. Isso implica que registos de transações sejam armazenados em outros servidores para que, em caso de problema ou avaria, seja possível restaurar as alterações levadas a cabo por transações bem-sucedidas (McCreary e Kelly, 2014).

3.3 Teorema CAP

No ano 2000, *Eric Brewer* apresentou a conjetura (Brewer, 2000) de que, em sistemas distribuídos, existe um compromisso fundamental entre consistência, disponibilidade e tolerância a partição. Esta conjetura foi em 2002 formalizada por *Seth Gilbert* e *Nancy Lynch* (Gilbert e Lynch, 2002) e tem, desde então, vindo a ser referida pelo meio académico como o teorema *CAP* (Benefico et al., 2012). Seguidamente serão expostos os 3 pontos no qual assenta este teorema:

- **Consistência**
  No contexto do teorema *CAP*, a consistência alude a atomicidade e isolamento. De uma forma simples, isso implica que todos os processos executados de forma concorrente visualizem a mesma versão dos dados (Tiwari, 2011).

- **Disponibilidade** (***A****vailability*)
  Este ponto significa que o sistema está disponível quando é solicitado. No contexto de um servidor *web*, isso significa que cada pedido eventualmente receberá uma resposta. Segundo o teorema *CAP*, o facto de a resposta não ser imediata pode representar um problema (Gilbert e Lynch, 2012).

- **Tolerância a partição** (***P****artition tolerance*)
  Este ponto implica que o sistema seja capaz de funcionar corretamente, mesmo no caso de falha por parte de algum dos componentes. A comunicação entre servidores *web* não é 100% fiável. Caso haja falha de comunicação entre servidores, várias máquinas podem ficar particionadas em grupos isolados, não podendo assim sincronizar dados entre si (Gilbert e Lynch, 2012).

O teorema *CAP* indica que qualquer sistema que suporte bases de dados distribuídas, apenas consegue em qualquer momento responder a 2 destes pontos simultaneamente (Mohanty et al., 2013). Este teorema pode ser explicado de uma forma simples:

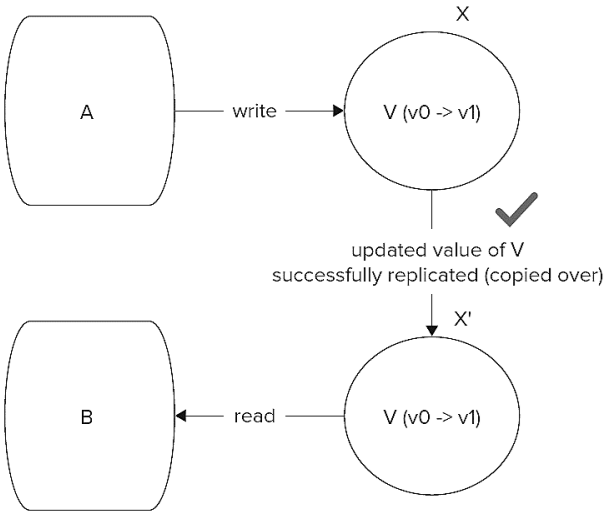

Figura 6 – Demonstração teorema CAP (1). Retirado de (Tiwari, 2011)

A Figura 6 ilustra uma situação com 2 nós de um sistema distribuído (*X* e *X'*). Na situação ilustrada existem dois processos (*A* e *B*) que interagem com os nós *X* e *X'* respetivamente. A referida figura demonstra que o processo *A* modifica o valor do atributo *V* de *v0* para *v1* e, como os 2 nós possuem um armazenamento de dados replicado, o nó *X* informa o nó *X'* da alteração ocorrida, *X'* atualiza o valor do atributo *V* de modo a que o processo *B*, quando requer uma leitura desse atributo, receba o valor já atualizado do mesmo (*v1*). Na iminência de existir um problema de rede e os dados não conseguirem ser sincronizados entre os 2 nós, o processo *B* vai receber o valor *v0*, que não é o valor do mesmo atributo, presente no nó *X* após a atualização. Nesta situação estaríamos perante um problema de consistência de dados.

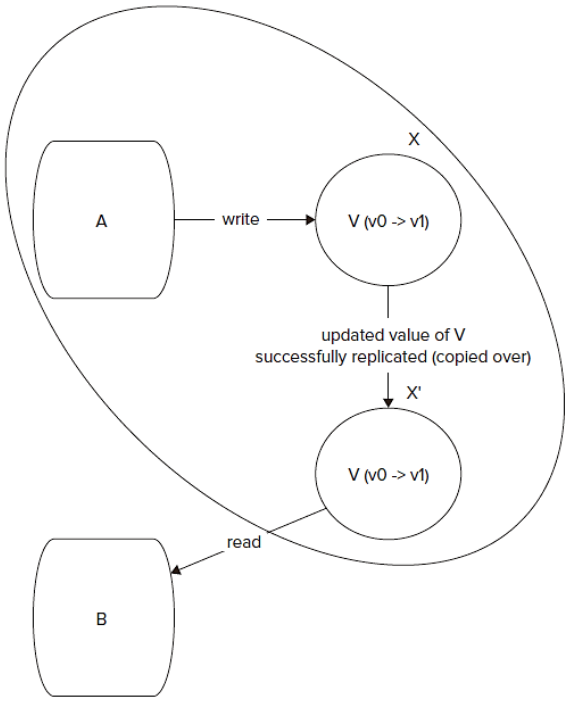

Figura 7 - Demonstração teorema CAP (2). Retirado de (Tiwari, 2011)

Na Figura 7 é apresentada uma solução para o problema de consistência exposto anteriormente. O esquema ilustrado sugere que a escrita dos dados por parte de *A* e a consequente sincronização entre os nós seja realizada numa única transação. Dada esta situação, na eventual ocorrência de uma falha de rede, o processo a ser levado acabo por *B*, seria bloqueado ou meramente cancelado até que a sincronização entre os nós se realize. Esta situação iria garantir a consistência mas, ao mesmo tempo, resultaria num impacto profundo na latência e disponibilidade do sistema. Este problema aplica-se independentemente de o processo de sincronização entre *X* e *X'* , ser síncrono ou assíncrono.

Se for seguido o teorema *CAP* existem então apenas 3 combinações possíveis:

- **Consistência e tolerância a partição, comprometendo a disponibilidade** (***CP***)
  Na situação em que um nó falha, o sistema pode ficar indisponível por bastante tempo, até que o sistema seja restaurado a um estado consistente. Este é um sistema típico onde transações monetárias e o tempo têm um papel importante. Em muitas situações, os sistemas conseguem recuperar e replicar rapidamente informação, levando a que o sistema esteja indisponível por um curto espaço de tempo. Na maioria dos casos, uma pequena quebra na disponibilidade não é catastrófica e revela-se uma opção viável (Tiwari, 2011).

- **Consistência e disponibilidade, comprometendo a tolerância a partição** (***CA***)
  Parte da base de dados não apresenta a preocupação de ser tolerante a falha e recorre, maioritariamente, à técnica de replicação para garantir a disponibilidade e consistência da informação. Normalmente, esta situação é encontrada nas tradicionais bases de dados relacionais (Han et al., 2011).

- **Tolerância a partição e disponibilidade, comprometendo a consistência** (***AP***)
  Em algumas situações, a disponibilidade não pode ser comprometida e o sistema é tão largamente distribuído que também não é possível comprometer a tolerância a partição. Nestes sistemas, no entanto, é possível abdicar de uma forte consistência. O termo fraca consistência não tem um significado bem definido. Pode variar entre nenhuma consistência e uma eventual consistência. Uma eventual consistência significa que, após a atualização de um atributo, "eventualmente" todos os nós do sistema irão sincronizar essa informação (Tiwari, 2011). Esta é uma das bases para o modelo *BASE*, apresentado na secção seguinte desta dissertação.

3.4 Propriedades BASE

O termo *BASE* foi criado pelos seguidores de *Eric Brewer*, na sequência da apresentação do teorema *CAP* (Tiwari, 2011), abordado na secção anterior. Como a própria palavra sugere, a origem da mesma deve-se à intenção dos seus criadores de salientar que as propriedades *BASE* (base) são filosoficamente opostas às propriedades *ACID* (ácido). Ao contrário do que o nome e o jogo de palavras possam transparecer, a grande diferença entre as duas não se situa na

escala do *Ph*, mas sim no espectro da consistência que os sistemas devem oferecer. Os sistemas *NoSQL* são amplamente baseados nas propriedades *BASE* (Bernardino e Abramova, 2013).

"*ACID* é pessimista e força a consistência no final de cada transação, *BASE* é otimista e aceita que a consistência da base de dados irá estar num estado de fluxo" (Pritchett, 2008). Ao contrário dos sistemas *ACID* que se focam na consistência, os sistemas *BASE* focam-se na disponibilidade. As propriedades *BASE* estão intrinsecamente associadas aos sistemas *NoSQL*, porque o principal objetivo destas propriedades é garantir que novos dados chegam a cada momento e que os mesmos necessitam de armazenamento imediato, mesmo que isso implique o risco de o sistema ficar dessincronizado por um breve período de tempo (McCreary e Kelly, 2014). Estes sistemas "relaxam" as regras para assim possibilitarem que *queries* sejam executadas, mesmo que nem todas as partes da base de dados em causa se encontrem devidamente sincronizadas. Os sistemas *BASE* são normalmente mais rápidos e apresentam uma estrutura mais simples; não há a necessidade de escrever código relativo a bloqueios e desbloqueios de recursos. O objetivo a atingir com esta filosofia passa por manter todos os processos em movimento e tratar das partes incompletas ou que falham, num momento posterior, quando o mesmo não colocar em causa a disponibilidade do sistema (McCreary e Kelly, 2014).

BASE significa ***B**asically **A**vailable*, ***S**oft-state*, ***E**ventually consistent*:

- **Basicamente disponível** (***B**asically **A**vailable*) significa que o sistema garante a disponibilidade dos dados de acordo com as propriedades do teorema *CAP*. No entanto, a resposta obtida pode ser "Falha" ou "Inconsistência", se os dados requeridos se encontrarem inconsistentes ou num estado de mudança (Celko, 2014).

- **Estado flexível** (***S**oft state*) significa que os dados acedidos podem estar incorretos ou imprecisos durante um variável período de tempo, e que esses dados podem estar a mudar ao mesmo tempo que são utilizados (McCreary e Kelly, 2014). Mesmo após grandes períodos de tempo sem novas adições de dados, não está garantida a total consistência dos mesmos, isto deve-se às implicações relativas ao ponto da "eventual consistência" (Celko, 2014). Em suma, a consistência dos dados não pode ser garantida (Bernardino e Abramova, 2013).

- **Eventual consistência** (***E**ventual consistency*) significa que o sistema irá eventualmente tornar-se consistente, uma vez cessadas todas as adições de dados a qualquer base de dados do sistema (Celko, 2014). Eventual consistência é um dos modelos de consistência utilizados no âmbito da programação paralela. Este modelo indica que, após um longo período de tempo sem alterações, todas as atualizações efetuadas sobre os dados de um nó do sistema irão ser propagadas a todos os outros pontos do sistema, e que todas as réplicas da base de dados irão eventualmente atingir um estado de consistência (Wang e Tang, 2012).

3.5 Taxonomia das Bases de dados *NoSQL*

Segundo a informação disposta em (Edlich, 2014) existem mais de 150 implementações distintas do paradigma *NoSQL*. Contudo, não existe uma classificação ou taxonomia oficial para estes sistemas (Tudorica e Bucur, 2011). Desde que o movimento *NoSQL* começou a ganhar tração, várias categorizações e classificações foram surgindo, tendo por base características como (Dharmasiri e Goonetillake, 2013):

- O Modelo de dados. Vários modelos de dados são utilizados pelos diferentes sistemas NoSQL. Por vezes, dentro de um modelo de dados existem várias implementações distintas.
- Métodos e linguagens utilizadas para se realizarem consultas sobre os dados.
- Os pontos do teorema *CAP* suportados pelo sistema.
- Modelo de consistência implementado.
- Características relacionadas com o modelo de integridade, indexação e distribuição dos dados (Moniruzzaman e Hossain, 2013).

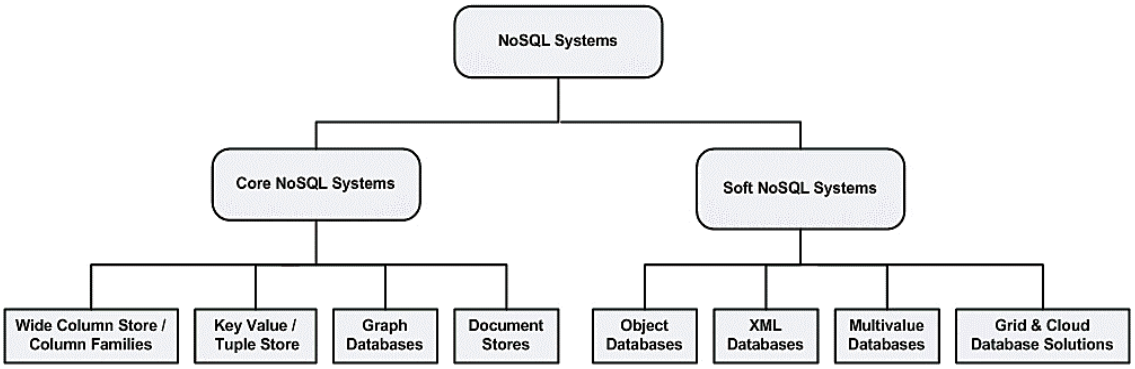

Figura 8 – Classificação dos sistemas NoSQL. Retirado de (Dharmasiri e Goonetillake, 2013)

Hoje, no entanto, o sistema taxonómico mais adotado divide as bases de dados *NoSQL* em 2 grandes grupos, "*Core NoSQL*" e "*Soft NoSQL*" (Dharmasiri e Goonetillake, 2013) (Edlich, 2014) (Tudorica e Bucur, 2011), como pode ser observado na Figura 8. Esta divisão deve-se ao facto de os sistemas "*Core NoSQL*" existirem como componentes de sistemas ou serviços, desenhados para suprir necessidades originadas pela *Web 2.0* (Tudorica e Bucur, 2011). Os sistemas "*Soft NoSQL*" por sua vez não partilham as ligações à *Web 2.0*, mas possuem algumas das características associadas ao movimento *NoSQL*, como por exemplo, a não utilização das propriedades *ACID*. Nesta dissertação serão apenas abordados e investigados os sistemas "*Core NoSQL*".

Dentro dos sistemas "*Core NoSQL*" existem também vários tipos de taxonomia diferentes, no entanto, a versão mais adotada é a ilustrada na Figura 9 (Bernardino e Abramova, 2013) (Hecht e Jablonski, 2011) (DBMS2, 2010) e que será analisada de seguida:

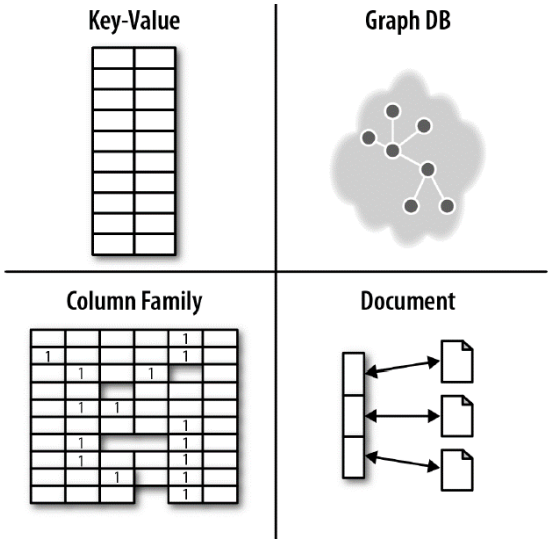

Figura 9 – As diferentes estruturas dos sistemas "Core NoSQL". Retirado de (Robinson et al., 2013)

3.5.1 Chave-Valor

A ideia de um armazenamento chave-valor, existe num contexto computacional há várias décadas. É um conceito ou estrutura de dados utilizada de forma bastante comum no desenvolvimento de sistemas de ficheiros (Indrawan-Santiago, 2012) ou, por vezes, como uma simples estrutura de dados. Os dados armazenados neste tipo de modelo, são constituídos por duas partes: uma *string* que representa a chave, e os dados a serem armazenados *per se*, que representam o valor, assim criando um par "chave-valor" (Nayak et al., 2013). Não existe nenhum requerimento específico para os dados a utilizar, estes podem possuir o tamanho desejado e podem ser representados por qualquer tipo de ficheiro (Celko, 2014), o que inclui poderem também representar outros conjuntos de chaves (Bernardino e Abramova, 2013). As chaves, por sua vez, podem possuir nomenclaturas bastante flexíveis, de acordo com as necessidades do sistema, como pode ser observado na Tabela 1.

Os sistemas de armazenamento do tipo chave-valor não possuem nenhuma linguagem destinada à realização de *queries*. Estes sistemas limitam-se a fornecer uma API (Nayak et al., 2013), que possui métodos que possibilitam a adição ou remoção de dados da base de dados (McCreary e Kelly, 2014). As limitações no que toca à profundidade (ou falta dela) das *queries* sobre a base de dados acabam por limitar bastante o sistema. Apesar de simples, este modelo não permite obter uma perceção total das informações e conhecimento contidos nos dados armazenados. Portanto, é necessário recorrer a uma estrutura de processamento externo, como o *Hadoop MapReduce*, para se conseguir alcançar uma melhor visão sobre o potencial dos dados (Robinson et al., 2013). Estas operações externas causam um elevado valor de latência, quando comparadas com operações de consulta semelhantes realizadas em outros tipos de base de dados.

Tabela 1 – Exemplos de pares chave-valor. Adaptado de (McCreary e Kelly, 2014)

Descrição da chave	Chave	Valor
Nome da imagem	image-12345.jpg	Ficheiro de imagem
URL de uma página Web	http://www.example.com/mywebpage.html	Página *HTML*
Caminho para um ficheiro	N:/folder/subfolder/myfile.pdf	Ficheiro PDF
Hash MD5	9e107d9d372bb6826bd81d3542a419d6	*"The quick brown fox jumps over the lazy dog"*
Comando para chamar uma função REST	view-person?personid=12345&format=xml	<Person><id>12345</id.</Person>
Consulta SQL	SELECT PERSON FROM PEOPLE WHERE PID="12345"	<Person><id>12345</id.</Person>

Como já foi referido, as bases de dados do tipo de armazenamento chave-valor possuem uma longa história. Todavia a nova vida destes sistemas de base de dados, apresentada agora na era *Big Data*, tem a sua grande influência no sistema *Dynamo* apresentada pela *Amazon* no artigo (DeCandia et al., 2007). Estes sistemas estão vocacionados para garantir grandes quantidades de leituras, o que se traduz num grande poder de escalabilidade por parte destes sistemas, ao mesmo tempo que nunca colocam em causa a sua disponibilidade. Segundo (Kaur e Rani, 2013), estas são as mais populares bases de dados NoSQL que recorrem ao tipo de armazenamento chave-valor:

3.5.1.1 ***Riak*** (Basho, 2014)

Riak é uma base de dados distribuída, tolerante a falha, de armazenamento do tipo par chave-valor. É um sistema de fonte aberta desde 2009. Este sistema é distribuído pela empresa do *Massachusetts*, *Basho technologies*, encontra-se em produção desde 2008, é altamente baseado no sistema *Dynamo* apresentado pela *Amazon* e é implementado recorrendo à linguagem *Erlang* (Fink, 2012).

Objetos *Riak* podem ser procurados e armazenados em *JSON*, o que permite que cada objeto possua vários campos. Neste sistema objetos podem ser agrupados em "baldes", tal como nas coleções suportadas pelas bases de dados que recorrem ao armazenamento no formato de documentos (a analisar no subponto seguinte). O sistema *Riak* não suporta índices ou outros campos de identificação para além da chave primária. Assim, acaba por não fornecer a mesma profundidade analítica que outros sistemas providenciam (Cattell, 2011).

Um grande ponto a favor deste sistema é a sua grande capacidade de tolerância a falhas. Servidores podem ser ligados, desligados ou falharem a qualquer momento, sem que tal afete

de qualquer modo o sistema (Redmond e Wilson, 2012). Um nó que falha não representa um problema de resolução imediata e pode ser tratado apenas quando existir disponibilidade para tal. O sistema *Riak* possui várias características, das quais se salientam as seguintes (Hurwitz et al., 2013):

- **Processamento Paralelo:** Recorrendo à utilização de *MapReduce*, o sistema apresenta capacidade para compor e decompor consultas transversalmente em qualquer conjunto de servidores, para assim conseguir alcançar a computação e análise em tempo real dos dados armazenados.
- **Ligações:** Um sistema *Riak* pode ser construído para "imitar" parte do comportamento de uma base de dados *NoSQL* de armazenamento do tipo grafo, para isso recorrendo à utilização de "*links*". Um "*link*" pode ser considerado como uma conexão unilateral entre dois pares chave valor. Seguir essas conexões irá permitir criar um mapa das relações existentes entre os vários pares chave-valor de uma base de dados.
- **Pesquisa:** O sistema *Riak* possui um método de pesquisa distribuído e tolerante a falhas que possui a capacidade de realizar análises completas sobre texto. O sistema ainda pode recorrer aos seus "baldes" para providenciar um método de transformação mais rápida de chaves em valores.

*3.5.1.2 **Redis*** (Redis, 2014)

Redis, que significa ***Re**mote **Di**ctionary **S**ervice* é uma base de dados *NoSQL* de armazenamento do tipo par chave-valor. Este sistema foi criado pelo programador italiano *Salvatore Sanfillipo*, que posteriormente foi contratado pela empresa *VMware*[7] para trabalhar no projeto a tempo inteiro (Lerner, 2010b). Este projeto, lançado em 2009, foi implementado recorrendo à linguagem *C* e é um projeto de fonte aberta (Cattell, 2011).

Este sistema trabalha em memória, oferecendo assim uma grande performance. Ao mesmo tempo, também fornece uma boa capacidade de replicação e um modelo de dados único para a construção de plataformas/aplicações orientadas para a resolução de problemas (Carlson, 2013). Este sistema tem a seu favor a facilidade de utilização e um conjunto sofisticado de comandos para interação com o sistema. Para além destes fatores, o sistema Redis é especialmente renomeado pela sua rapidez. De acordos com indicadores obtidos em testes de performance, as leituras neste sistema são bastante rápidas e as escritas ainda mais; o sistema é capaz de tratar e gerir mais de cem mil operações por segundo (Redmond e Wilson, 2012). Esta velocidade deve-se, em grande parte, ao facto de este sistema manter o conjunto de dados a utilizar em memória.

Este sistema é também capaz de oferecer persistência de dados, porque possui um mecanismo que, de uma forma assíncrona, escreve as alterações realizadas, em disco (Haines, 2009). É possível configurar de quanto em quanto tempo, ou depois de quantas transações é que o sistema deve guardar os dados. Deste modo, se um servidor falhar, será grande a possibilidade

[7] VMware é uma organização norte-americana especializada no fornecimento de *software*/serviços de virtualização e computação em nuvem.

de não serem perdidas várias transações ou modificações realizadas sobre os dados. O ponto fraco do sistema *Redis* é o facto de que o tamanho do conjunto de dados a utilizar tem de caber confortavelmente na memória *RAM* disponível no sistema (Haines, 2009), estando limitado nesse aspeto em relação a outras soluções *NoSQL*, e mais especificamente, a outros sistemas de armazenamento do tipo par chave-valor.

3.5.1.3 ***Voldemort*** (Project-Voldemort, 2014)

À semelhança dos dois sistemas que o precedem nesta lista, o sistema *Voldemort* é um projecto *NoSQL* de uma base de dados distribuída do tipo par chave-valor. É altamente baseada no sistema *Dynamo* apresentado pela Amazon, e utiliza um particionamento baseado em *Hash* para distribuir os dados pelos vários nós de armazenamento (Pirzadeh et al., 2011). Este projeto foi inicialmente desenvolvido no *LinkdIn* em 2008 e a sua criação tinha por objetivo fornecer suporte do tipo chave-valor para armazenamento de dados do tipo "Quem viu o meu perfil" (Auradkar et al., 2012). Tornou-se de fonte aberta em 2009 e desde então tem vindo a ser adotado por várias organizações e projetos, que requerem um armazenamento com um alto nível de disponibilidade e ao mesmo tempo pouca latência.

Este sistema provisiona um mecanismo de controlo de concorrências multi-versão (MVCC), as atualizações realizadas às réplicas da base de dados presentes em outros nós, são realizadas de forma assíncrona. Assim, este sistema não consegue garantir a consistência dos dados em todos os momentos (Cattell, 2011), regendo-se essencialmente pelo ponto da eventual consistência do teorema *BASE*. Devido à política de particionamento baseada em *Hash*, este sistema não apresenta suporte nativo para consultas baseadas em intervalos de dados (Cattell, 2011). O sistema *Voldemort* acaba por ser bastante flexível, de modo a que suporta a ligação de outros motores de armazenamento ao sistema, como por exemplo o sistema *BerkleyDB* ou *MySQL*.

3.5.1.4 DynamoDB (Amazon, 2014a)

Este sistema não é considerado pela fonte citada (Kaur e Rani, 2013) como um dos mais populares na área das bases de dados *NoSQL* de tipo de armazenamento chave-valor, mas irá ser analisado brevemente nesta dissertação, devido à grande influência que o sistema (que o precedeu) teve no movimento *NoSQL* em geral, e nos sistemas de armazenamento do tipo par chave-valor em particular.

Como já foi referido nesta dissertação, em 2007 a *Amazon* apresentou no artigo (DeCandia et al., 2007), o seu sistema de armazenamento do tipo chave-valor que tinha como objetivo suportar a secção do "carrinho de compras" da sua loja *online*. Este sistema precisava de ser altamente disponível e fiável, teria de estar preparado para funcionar 24 horas por dia, 7 dias por semana sem falhas e sem afetar o processo de compra por parte de clientes (McCreary e Kelly, 2014). Citando diretamente o artigo da *Amazon*: "*customers should be able to view and add items to their shopping cart even if disks are failing, network routers are flapping, or data centers are being destroyed by tornados*". No entanto, este sistema nunca foi disponibilizado ao público, sendo apenas utilizado internamente na *Amazon.* Não obstante esta situação, o artigo publicado teve um grande impacto no movimento *NoSQL*. Os 3 sistemas apresentados anteriormente possuem todos grandes bases filosóficas neste artigo.

Em 2012, a *Amazon* decidiu disponibilizar um novo sistema, baseado no *Dynamo*. Esse sistema tomou o nome de *DynamoDB*. Os dois sistemas possuem algumas diferenças arquitetónicas, mas as bases são as mesmas. O *DynamoDB* é um sistema de gestão de bases de dados *NoSQL* do tipo par chave-valor, altamente tolerante a falhas, com foco na disponibilidade, na escalabilidade transparente e na simplicidade de utilização (Hurwitz et al., 2013). Este serviço encontra-se disponível para utilização, mediante o pagamento de uma subscrição, na loja de serviços *web*, da *Amazon* (*AWS)*[8].

3.5.2 Documento

Este tipo de bases de dados é considerado como o mais generalista, flexível, poderoso e popular de todos os que integram o movimento *NoSQL* (McCreary e Kelly, 2014). Como o próprio nome indica, este tipo de bases de dados recorre a documentos como método de armazenamento de dados, quase como um armário ou arquivo, destinado a armazenar ficheiros da era digital. Este tipo de bases de dados fornece uma boa performance ao mesmo tempo que possibilita uma grande escalabilidade horizontal (Nayak et al., 2013). Os documentos armazenados são normalmente de tipos bastante comuns como *XML*, *JSON* ou *PDF* (Nayak et al., 2013).

Neste tipo de sistemas a cada documento é atribuída uma chave única. Estas chaves podem ser representadas por uma simples *string*, por um caminho de ficheiro (Nayak et al., 2013) ou um outro tipo de *URL* ou *URI*. De uma maneira geral estes sistemas baseiam-se em índices para tornar mais fácil o acesso a documentos (Robinson et al., 2013). Uma consequência da utilização de documentos como armazenamento é que, sempre que um novo documento é guardado, todo o conteúdo desse documento tem de ser indexado (McCreary e Kelly, 2014).

Como nos índices das bases de dados relacionais, índices neste tipo de bases de dados permitem trocar a performance de escrita, por uma maior performance de leitura. No entanto, os índices destes sistemas são bastante amplos, o que significa que todos os critérios são pesquisáveis (McCreary e Kelly, 2014). Basta conhecer uma propriedade de um documento, para assim conseguir encontrar rapidamente a mesma propriedade em outros documentos. Os sistemas de armazenamento do tipo chave-valor conseguem armazenar um documento inteiro como o valor associado a uma chave. Mas uma base de dados com armazenamento do tipo documento consegue rapidamente extrair secções de documentos, sem ter que carregar documentos inteiros para memória (McCreary e Kelly, 2014).

Documentos são agrupados em coleções (Kaur e Rani, 2013). Se for realizado um paralelismo com as bases de dados relacionais, podemos considerar que as tabelas do modelo relacional, são o equivalente às coleções do modelo de armazenamento de documentos, e os registos de cada tabela são equivalentes aos documentos de cada coleção. Estas bases de dados são bastante flexíveis por contraposto ao modelo relacional. Isto porque documentos diferentes de uma mesma coleção podem ter diferentes números de campos ou propriedades. Ao contrário

[8] Coleção de serviços de computação remota que juntos formam uma plataforma de computação em nuvem, disponibilizada pela *Amazon* através da internet e acessível em http://aws.amazon.com

do que sucede numa base de dados relacional, não é necessário gastar espaço em disco a adicionar campos em branco a todos os outros documentos de uma coleção (Kaur e Rani, 2013). Outra diferença para o modelo relacional reside no facto de que uma tabela não pode possuir numa das suas células uma outra tabela. No caso do armazenamento de documentos, no entanto, é possível criar coleções, que são armazenadas dentro de uma coleção (McCreary e Kelly, 2014).

As bases de dados *NoSQL* do tipo armazenamento de documentos são as mais indicadas para aplicações *web* que necessitem de armazenar grande quantidade de dados semiestruturados, onde também é necessário executar várias consultas dinâmicas (Kaur e Rani, 2013). Segundo (Kaur e Rani, 2013) os sistemas *NoSQL* mais adotados que recorrem ao modelo de armazenamento do tipo documento são:

3.5.2.1 ***MongoDB*** (MongoDB, 2014i)

No contexto das bases de dados *NoSQL* é difícil de bater a facilidade de utilização oferecida pelo sistema *MongoDB*. Não é só extremamente bem documentado, como ainda é suportado por uma grande e prestável comunidade. Ao mesmo tempo revela-se bastante amigável para programadores que possuam experiência com sistemas relacionais e linguagem *SQL*, tornando-o num sistema especialmente indicado para novatos no mundo *NoSQL* (Wilson, 2013).

MongoDB é um sistema *NoSQL*, orientado para o armazenamento sob a forma de documentos e desenvolvido em C++. O seu nome advém da palavra anglo-saxónica "*Hu**mongo**us*", normalmente utilizada para qualificar algo gigantesco. O suporte comercial para este sistema é fornecido pela organização *10Gen*, que iniciou o seu desenvolvimento no último trimestre de 2007 (Liu et al., 2012).

Documentos neste sistema são serializados naturalmente no formato *JSON* (*Javascrip Object Notation)* e estes documentos são armazenados fisicamente recorrendo à codificação binária do formato *JSON*, denominada de *BSON* (Dede et al., 2013). Os ficheiros *BSON* geridos pelo sistema não podem ultrapassar um tamanho máximo de 19 Megabytes (Bernardino e Abramova, 2013). Neste sistema os documentos são agrupados em coleções, em conformidade com a sua estrutura. Alguns documentos com diferentes estruturas também podem ser armazenados na mesma coleção mas, para manter uma boa performance, é aconselhável agrupar, na mesma coleção, documentos com estruturas similares (Bernardino e Abramova, 2013). De modo a aumentar a performance do sistema, ficheiros de dados são mapeados em memória. Por defeito esses dados são enviados para disco a cada 60 segundos, mas esse valor pode ser modificado. Quando novos ficheiros são criados, todos os dados são imediatamente enviados para disco, assim libertando memória (Bernardino e Abramova, 2013).

O sistema *MongoBD* utiliza um sistema de indexação parecido com aquela a que recorrem as bases de dados relacionais. Cada documento é identificado por um campo "_id", nesse campo é criado um índice único. A indexação é um processo importante para a eficiência das operações de leitura, no entanto isso pode revelar-se como um fator que impacta negativamente a performance de operações de inserção de dados. Outros índices podem ser criados pelo administrador do sistema (Bernardino e Abramova, 2013).

Dois dos pontos mais relevantes deste sistema são: a forma como gere o mecanismo de concorrência e os processos que utiliza para garantir a durabilidade dos dados. A durabilidade dos dados é garantida pela criação de réplicas. O sistema *MongoDB* utiliza um mecanismo de replicação mestre-escravo. Assim o utilizador define um nó mestre, e um ou vários nós escravos. O mestre pode ler ou escrever ficheiros. Os nós escravos estão limitados a servir como reservas, assim apenas operações de leitura são permitidas nesses nós. Se o nó mestre falhar, o nó escravo que possua a réplica de dados mais recente é diligenciado como mestre. Todas as réplicas são atualizadas assincronamente, assim alterações aos dados não são propagadas imediatamente (Bernardino e Abramova, 2013). A consistência dos dados apresentada por este sistema, enquadra-se no nível de "eventual consistência" inerente aos sistemas *BASE*.

O sistema *MongoDB* oferece nativamente vários controladores, que permitem que interações com o sistema, sejam realizadas por aplicações desenvolvidas em várias linguagens, tais como *C*, *C++, Java*, *.NET*, *JavaScript* e *PHP.* Existem ainda várias *frameworks* e bibliotecas disponibilizadas por terceiros que permitem interagir com o sistema de várias formas em diferentes linguagens (Sattar et al., 2013). Muitas das grandes organizações a nível mundial, possuem nos seus repositórios de dados, um sistema *MongoDB*.

*3.5.2.2 **CouchDB*** (Apache, 2014a)

O sistema *couchDB*, cuja sigla significa "***C**luster **O**f **U**nreliable **C**ommodity **H**ardware*", foi criado por *Damien Katz* e disponibilizado em 2005 como um projeto de fonte aberta (McCreary e Kelly, 2014). Este sistema é uma base de dados vocacionada para o armazenamento de dados sob a forma de documentos e foi escrito em *Erlang* (Chen et al., 2014). Este tem vindo a integrar a lista de projetos *Apache* desde meados de 2008. Os dados neste sistema são organizados em documentos constituídos por campos do tipo par chave-valor, que são armazenados e acedidos como objetos *JSON* (Chen et al., 2014). Cada documento possui um identificador único. As coleções de documentos são o único esquema de base de dados presentes neste sistema e índices secundários têm de ser explicitamente criados em campos dentro das coleções (Cattell, 2011).

O *ChouchDB* foi pensado de raiz para lidar com a *web* e por consequência também com os seus defeitos e falhas. Este sistema oferece um nível de robustez que não consegue ser equiparado por outras bases de dados. Outros sistemas toleram falhas de rede, mas o *CouchDB* consegue operar mesmo quando a ligação aos outros nós da rede está raramente disponível (Redmond e Wilson, 2012).

As consultas neste sistema são realizadas através de funções *MapReduce* escritas em *JavaScript*, que no contexto *CouchDB* tomam a designação de "vistas" (Warden, 2011). Esta abordagem torna mais fácil para o sistema gerir o processamento de dados de uma forma distribuída. As "vistas" oferecem grande poder e flexibilidade mas podem apresentar uma complexidade elevada, mesmo quando se trata de uma consulta bastante simples (Warden, 2011).

*3.5.2.3 **RavenDB*** (HibernatingRhinos, 2014)

RavenDB é um sistema *NoSQL* de armazenamento do tipo documento, originado em 2010. É um sistema de fonte aberta, especialmente desenvolvido para operar no sistema operativo *Windows*, e com aplicações que recorrem ao ecossistema *.NET* (Ritchie, 2013).

Este sistema suporta múltiplas bases de dados, uma base de dados é regida pelo sistema como se este fosse um contentor de dados. Assim este sistema consegue facilmente gerir centenas de milhares de bases de dados na mesma instância. O sistema é capaz de gerir várias bases de dados, mas num qualquer dado momento, apenas uma instância está ativa e a consumir recursos do sistema (Tannir, 2013).

O sistema *RavenDB* possui índices, mas utiliza-os de um modo diferente daquele empregue pelas bases de dados relacionais. Índices são utilizados para satisfazer consultas. O sistema recebe e analisa a consulta e posteriormente extrai um índice que possui informação capaz de responder à consulta efetuada. O sistema é capaz de gerar automaticamente índices novos, que serão incluídos na base de dados, a partir das consultas realizadas (Tannir, 2013).

Documentos neste sistema são armazenados no formato *JSON*. O *RavenDB* não possui um esquema de bases de dados, os seus documentos não têm de conter em si um campo ou coluna específico. Neste sistema também não existe o conceito de relações entre documentos como existe entre as tabelas de uma base de dados relacional. No entanto, é possível criar ligações entre documentos, mas estas ligações existem apenas nos dados, e não são impostas pelo esquema da base de dados. Este sistema é capaz de armazenar grandes quantidades de dados e também é capaz de otimizar automaticamente a maneira como os documentos são armazenados e geridos. O sistema consegue escalar horizontalmente com facilidade e cada documento pode ser armazenado num qualquer nó de armazenamento que faça parte da rede de recursos (Tannir, 2013).

3.5.3 Família de Colunas

Os sistemas *NoSQL* do tipo família de colunas são conhecidos por possuírem uma capacidade notável de escalar horizontalmente, de modo a conseguirem albergar grandes quantidades de dados (McCreary e Kelly, 2014). Estes sistemas também são conhecidos por estarem intimamente relacionados com vários sistemas que recorrem a funções *MapReduce*. Quase todos os sistemas deste tipo em existência são altamente influenciados pelo artigo (Chang et al., 2006), onde em 2006 a *Google* apresentava o *BigTable*: um sistema de armazenamento distribuído para gerir dados estruturados, desenhado com o intuito de conseguir escalar horizontalmente de forma natural, suportando quantidades de dados muito elevadas (Hecht e Jablonski, 2011).

Devido ao seu formato tabular, estes sistemas apresentam algumas semelhanças visuais com as bases de dados relacionais. A grande diferença entre os dois está na maneira como ambos gerem os valores nulos. Numa situação em que é necessário armazenar um grande número de atributos, um sistema relacional irá guardar um valor nulo para cada coluna em que não existam

dados a guardar. Por contraposto, num sistema *NoSQL* de armazenamento família de colunas irão apenas ser armazenados pares chaves-valor numa linha, se esses dados existirem e forem necessários (Hecht e Jablonski, 2011). Este tipo de sistemas *NoSQL* é muitas vezes considerado como um tipo específico do modelo par chave-valor. Os sistemas de bases de dados que recorrem a este modelo definem a estrutura do valor (no par chave-valor) como um conjunto predefinido de colunas (Indrawan-Santiago, 2012). De um modo pragmático, podemos encarar o modelo de dados destes sistemas como sendo uma tabela de baixa densidade, cujas linhas podem conter um número arbitrário de colunas. As chaves de cada linha (ou par chave-valor) fornecem um meio de indexação natural (Robinson et al., 2013). As estruturas de armazenamento do tipo família de colunas possuem 4 componentes principais utilizados na sua estrutura: colunas, super colunas, família de colunas e super família de colunas (Robinson et al., 2013).

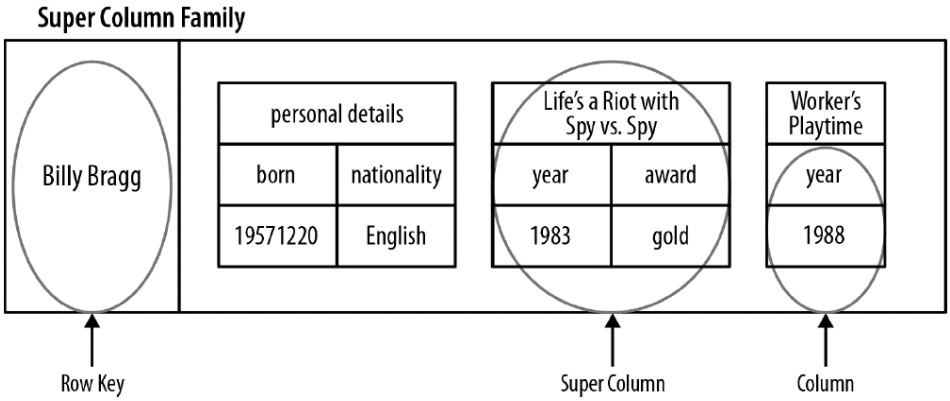

Figura 10 - Componentes estruturais do modelo família de colunas. Retirado de (Robinson et al., 2013)

Como pode ser observado na figura anterior, a coluna representa a estrutura mais elementar deste modelo e é constituído por um par nome-valor. Qualquer número de colunas pode ser combinado para gerar uma super coluna. Colunas e super colunas são por sua vez armazenadas em filas; quando uma fila possui apenas colunas, obtém a designação de família de colunas; se possuir na sua estrutura super colunas então a designação tomada é a de super família de colunas (Robinson et al., 2013). Como referido anteriormente, as chaves neste modelo fornecem um meio de indexação natural. Essa situação pode ser observada na Figura 10; as várias colunas contidas na linha possuem informação relativa ao músico *Billy Bragg*, desse modo essa é a *string* atribuída como chave dessa linha. A estrutura da família de colunas ou da super família de colunas determina o esquema da base de dados. Mas essa estrutura não é fixa, ao contrário das bases de dados relacionais, nestes sistemas, colunas podem ser adicionadas depois da base de dados se encontrar em utilização. Segundo (Kaur e Rani, 2013) estes são os sistemas *NoSQL* mais populares, que recorrem a este modelo de dados:

3.5.3.1 ***HBase*** (Apache, 2014e)

O sistema *HBase* foi criado em 2007 por uma empresa de *San Francisco* denominada de *Powerset*. Este sistema foi desde a sua criação encarado como uma contribuição para o ecossistema *Hadoop,* previamente abordado na secção 2.3.2 desta dissertação. Em meados de 2008 a empresa que o originou foi comprada pela *Microsoft* e, consequentemente, o suporte

para o desenvolvimento deste projeto terminou (George, 2011). Desde então o sistema tem vindo a fazer parte dos projetos *Apache*, onde começou por ser um subprojecto *Hadoop* até mais tarde alcançar o estatuto de projeto de topo.

O *Apache HBase* é um projeto de fonte aberta que consiste numa base de dados *NoSQL* orientada para o armazenamento do tipo família de colunas, distribuída, tolerante a falhas e altamente escalável (Vora, 2011). Este sistema pode ser encarado como uma base de dados que está construída em cima do sistema *HDFS*, pertencente ao ecossistema *Hadoop* (Bakshi, 2012). Esta base de dados foi criada à semelhança do sistema *BigTable* apresentado pela *Google* em (Chang et al., 2006). Desta forma, grande parte das suas funcionalidades e implementações apresentam uma enorme semelhança com aquelas dispostas pelo *BigTable* (Konishetty et al., 2012). Este sistema é especialmente indicado para situações onde é necessário alojar tabelas com tamanhos bastante elevados, esparsas e distribuídas por um grande número de máquinas. Estas tabelas podem conter biliões de linhas e milhões de colunas (Konishetty et al., 2012).

Neste sistema, os dados são armazenados em tabelas; cada tabela contem múltiplas linhas, mas um número fixo de colunas. Para cada linha podem existir vários "qualificadores" dentro de uma mesma família de colunas. Na interseção entre "qualificadores" e linhas encontram-se as células das tabelas. Linhas são ordenadas por chaves de linha, que são implementas como *arrays* de *bytes* (Gao et al., 2011). Este sistema, quando armazena em disco os dados relativos a uma família de colunas, fá-lo de modo a estes ficarem fisicamente agrupados, e que os itens de uma certa coluna possuam todos as mesmas características de escrita/leitura e os mesmos tipos de dados. Apenas uma linha pode ser bloqueada por defeito a um qualquer dado momento. A escrita de dados é um processo sempre atómico, mas uma linha pode ser bloqueada para conseguir alcançar uma operação de leitura e escrita ao mesmo tempo (Carstoiu et al., 2010).

O sistema não possui nos seus elementos principais uma linguagem declarativa para realização de *queries*. O *Hbase* lida com falhas de forma completamente transparente para o utilizador. A escalabilidade é completamente integrada no sistema. Novos nós podem ser adicionados aos recursos disponíveis em tempo real, sem necessidade de o utilizador realizar processos de redistribuição ou de rebalanceamento. O sistema realiza essas tarefas automaticamente (George, 2011). Este sistema é ativamente utilizado e desenvolvido por várias organizações importantes com necessidades amplas de gestão de dados *Big Data*, como o *Facebook*, *eBay* e *Yahoo!* (Redmond e Wilson, 2012).

*3.5.3.2 **Cassandra*** (Apache, 2014d)

Nos primeiros anos da sua atividade, o *Facebook* utilizava uma base de dados comercializada por terceiros na sua arquitetura interna em conjunção com *Hadoop*. O crescimento exponencial de utilizadores e o consequente aumento no número de dados a armazenar e analisar, levou a que surgisse a necessidade de pensar num sistema de armazenamento de escalabilidade ilimitado com foco na disponibilidade e distribuição. Em 2008, a equipa do *Facebook* construiu então uma arquitetura que combinava a abordagem do modelo de dados apresentada pelo sistema *BigTable* com a abordagem da infraestrutura do sistema *Dynamo* (Krishnan, 2013). Este sistema que foi apresentado no artigo (Lakshman e Malik, 2010) e tomou a designação de

Cassandra. Algum tempo mais tarde, o *Facebook* doou o código do sistema à fundação *Apache* e desde então é um projeto de fonte aberta. O *Facebook* não participa no desenvolvimento da versão de fonte aberta do *Cassandra*, mas ainda o utiliza nos seus sistemas (Lerner, 2010a).

O *Apache Cassandra* é um sistema de armazenamento *NoSQL*, desenvolvido em J*ava*. Sendo este um sistema que adota o tipo de armazenamento família de colunas, apresenta uma estrutura parecida com a de um sistema relacional tradicional (Bernardino e Abramova, 2013). Sendo este sistema também baseado no *BigTable*, o seu modelo de dados acaba por apresentar similaridades com o *HBase*. O sistema Cassandra foi desenhado para conseguir gerir grandes volumes de dados, alcançando simultaneamente uma grande disponibilidade e poder de escalabilidade, sem nenhum ponto de falha (Aniello et al., 2013). Como no sistema *HBase*, o *Cassandra* considera as falhas de hardware como normais e não como exceções. Assim o sistema está preparado para, automaticamente, compensar a perda de um nó sem que isto tenha qualquer significado para o utilizador final.

Este sistema é renomeado pela sua alta velocidade de escrita, sem que isso afete a eficiência das leituras de dados (Bagade et al., 2012). Como é uma base de dados distribuída, o sistema replica os dados para manter a latência de pesquisa baixa. Desse modo este sistema também é caracterizado pelo seu modelo, que relaxa as regras de consistência baseado em quóruns. Réplicas distintas da base de dados em causa podem possuir dados diferentes dependendo do momento temporal. Estes erros são prontamente detetados e as réplicas da base de dados reconciliam-se rapidamente, o que se deve aos rótulos temporais que acompanham cada processo de escrita de dados no sistema (Beernaert et al., 2013). Este sistema dá ao utilizador a flexibilidade de poder escolher o modelo de consistência que pretende; esta seleção varia entre a eventual consistência (característica dos sistemas *NoSQL*) e uma forte consistência (Aniello et al., 2013).

*3.5.3.3 **HyperTable*** (Hypertable, 2014)

À semelhança dos dois sistemas anteriores, também o *Hypertable* se baseia no sistema *BigTable* apresentado pela *Google*. Este sistema representa uma alternativa de alta performance ao *HBase*, sendo maioritariamente desenvolvido em *C++* e não em *Java* como os dois sistemas anteriores (Cattell, 2011). Este projeto de fonte aberta iniciado em 2007 (Tiwari, 2011) tem por objetivo fornecer um conjunto de sistemas de alta performance capazes de armazenar e processar dados de uma forma distribuída (Chen et al., 2014). Este requer a utilização subjacente de um sistema de ficheiros distribuído (Cattell, 2011), como por exemplo, o *HDFS* abordado na secção 2.3.2.

O sistema *Hypertable* possui um tipo de armazenamento bastante parecido com o do *HBase*. O tipo de armazenamento é baseado em famílias de colunas, e os dados são armazenados em chaves linha de uma forma distribuída e ordenada. No sistema *Hypertable* todas as informações relativas à versão dos dados são armazenadas conjuntamente com esses dados. Esta informação é identificada através de identificadores temporais (*timestamps*) (Tiwari, 2011).

Apesar de fornecer um armazenamento do tipo família de colunas, as características do seu armazenamento físico são afetadas pelo conceito de grupos de acesso. Grupos de acesso no

sistema *Hypertable* permitem que dados relacionados pertencentes a uma mesma coluna, sejam fisicamente armazenados, lado a lado. Nos sistemas relacionais tradicionais, os dados são distribuídos por linhas, e armazenados como tal. Isso significa que dados de duas linhas adjacentes são fisicamente armazenados, uns imediatamente ao lado dos outros. Já nos sistemas orientados para o armazenamento em colunas, dados de duas colunas adjacentes são fisicamente armazenados, uns imediatamente ao lado dos outros. Com os grupos de acesso, o sistema *Hypertable* possui a flexibilidade de colocar uma ou mais colunas no mesmo grupo. Manter todas as colunas no mesmo grupo de acesso, permite simular o ambiente de um sistema relacional. Manter cada coluna em grupos de acesso diferentes, permite simular um sistema de armazenamento orientado para colunas (Tiwari, 2011). Este sistema possui a sua própria linguagem orientada para a realização de *queries*, denominada de *Hypertable Query Language* (*HQL*). Esta linguagem permite aos utilizadores criarem, modificarem, eliminarem e consultarem dados em colunas pertencentes ao sistema (Chen et al., 2014).

3.5.4 Grafo

Vários estudos e pesquisas tendo por base modelos de dados baseados em grafos, têm vindo a ser realizados desde os anos 70 (Angles e Gutierrez, 2008). A atividade associada a este modelo e às bases de dados que nele se baseiam, teve o seu momento mais próspero na primeira metade dos anos 90 do século *XX*, seguido de um quase total abandono e esquecimento por parte do mundo académico e dos seus investigadores (Angles e Gutierrez, 2008). Mais recentemente com a introdução dos primeiros conceitos associados à *web 3.0* e aos projetos *Linked Data* (Berners-Lee et al., 2009), conjugado com a cada vez mais omnipresente cultura das redes sociais, voltou a trazer estas bases de dados para a linha da frente do interesse dos investigadores, quer a um nível académico quer a um nível empresarial.

Neste tipo de bases de dados, o modelo de dados representa uma rede que contém nós, arestas e propriedades (McCreary e Kelly, 2014). As arestas são utilizadas para ligar dois nós e representam uma relação, as relações podem possuir uma direção, conferindo assim um significado à relação. Os nós podem possuir propriedades, estas descrevem com variáveis graus de profundidade os dados contidos nesses nós; arestas também podem possui os seus próprios conjuntos de propriedades. As relações são identificadas pelos seus nomes e podem ser atravessadas em ambas as direções (Kaur e Rani, 2013).

Os sistemas de gestão de bases de dados relacionais utilizam números artificiais como chaves primárias e chaves estrangeiras, de modo a conseguir relacionar linhas que estão em tabelas armazenadas em sectores diferentes de um disco rígido. Operações de agregação de tabelas revelam-se bastante caras em termos de performance nestes sistemas, impulsionado por fatores como a latência e escritas/ leituras de disco (McCreary e Kelly, 2014). Ao contrário das bases de dados relacionais, as bases de dados que recorrem ao armazenamento to tipo grafo, possuem ferramentas de aglomeração de dados, computacionalmente bastante leves e rápidas. Essa situação deve-se à capacidade destes sistemas conseguirem manter grandes redes de grafos inteiras em memórias, evitando assim as pesadas, e normalmente mais lentas, operações

de disco (McCreary e Kelly, 2014). Ao contrário de outros tipos de bases de dados *NoSQL*, este tipo de bases de dados não possui as mesmas propriedades de escalabilidade por vários servidores. Os dados podem ser replicados por vários servidores de modo a melhorar a performance de leituras e consultas, no entanto escrever dados em vários servidores e realizar consultas a grafos que se estendem por vários nós em diferentes servidores podem revelar-se de complexa e difícil implementação (McCreary e Kelly, 2014). Segundo (Kaur e Rani, 2013) as soluções mais populares representativas deste tipo de bases de dados *NoSQL* são:

3.5.4.1 Neo4J (Neo-Technology, 2014a)

Uma base de dados *NoSQL,* disponibilizada pela empresa *Neo Technology,* esteve em desenvolvimento por uma década antes de a primeira versão alcançar o público em meados de 2010 (Nawroth, 2010). *Neo4J* é um sistema integrado, baseado em disco, totalmente transacional, desenvolvido em *Java,* que armazena dados em estruturas de grafos, em vez de tabelas (Vicknair et al., 2010). Este sistema é considerado o líder no âmbito das bases de dados do tipo grafo (Webber, 2012) e isso deve-se, em grande parte, ao seu modelo intuitivo e expressivo, que permite obter dados altamente relacionados. Este sistema é bastante mais rápido do que as bases de dados relacionais, tornando-o ideal para gerir dados complexos em vários domínios (Webber, 2012). Este sistema possui duas versões distintas: uma versão comunitária gratuita e de código aberto, bastante limitada em relação às outras versões, e uma versão empresarial, gratuita para utilizadores individuais, com limitações ao nível da utilização de recursos. Essa mesma versão possui também 2 outros módulos de pagamento aplicáveis, dependendo do tamanho da organização e dos recursos computacionais que a mesma pretenda utilizar. Segundo (Neo-Technology, 2014b) e (Hurwitz et al., 2013) estas são as características mais importantes do sistema *Neo4J*:

- **Intuitivo:** utiliza um modelo de grafos para representação dos dados.
- **Fiável:** Suporta transações ACID.
- **Rápido e resistente:** Utiliza um motor de armazenamento nativo baseado em disco. Possui uma *framework* transversal poderosa para a realização de consultas de alta velocidade.
- **Altamente disponível e massivamente escalável:** Quando distribuído por várias máquinas, o sistema pode alcançar milhares de milhões de nós e relações.
- **Linguagem de realização de *queries*:** Este sistema suporta uma linguagem declarativa denominada de Cypher, desenhada especificamente para realizar *queries* sobre grafos e os seus componentes. Esta linguagem é vagamente baseada na sintaxe *SQL*.

3.5.4.2 Titan (Aurelius, 2014c)

Este sistema é um dos mais recentes projetos a ver a luz do dia, no mundo das bases de dados *NoSQL* de armazenamento do tipo grafo; foi apresentado ao público em 2012. Como vários dos sistemas deste âmbito, é escrito em *Java* e é um projeto de fonte aberta. Este sistema é altamente escalável, otimizado para armazenar e realizar consultas sobre grafos que possuam milhares de milhões de nós e arestas, e que possam estar distribuídos por vários conjuntos de máquinas. É uma base de dados transacional capaz de suportar milhares de utilizadores a

realizarem travessias de grafos complexas de forma concorrente (Aurelius, 2014a) e possui uma arquitetura de armazenamento capaz de suportar ligações a outros tipos de bases de dados do tipo par chave-valor ou família de colunas. O sistema suporta ligações a 3 bases de dados em particular: *HBase*, *BerkelyDB* e *Cassandra*. A opção por uma das 3 bases de dados, influenciará de maneira contrastante as propriedades transacionais e de escalabilidade do sistema (Kolomicenko et al., 2013). Essa opção deve reger-se pelos requerimentos e pelas regras do negócio ou projeto em causa.

*3.5.4.3 **OrientDB*** (OrienTechnologies, 2014)

Este sistema representa uma base de dados *NoSQL* de fonte aberta apresentado em meados de 2010 e o seu ponto mais interessante é a sua arquitetura de armazenamento multimodelo. Assim providencia suporte e combinação de 3 tipos de armazenamento diferentes: par-chave valor, colunas e grafos (Kolomicenko et al., 2013). Esta situação permite, por exemplo, que documentos possuam arestas entre eles, assim emulando as propriedades de adjacência de documentos sem recurso a meios de indexação (Barmpis e Kolovos, 2012). As transações neste sistema apresentam suporte completo das propriedades *ACID*; é possível realizar escritas e leituras concorrentes no mesmo registo, sem necessidade de o sistema bloquear a base de dados. No entanto, todos os registos devem possuir a sua versão para que a idade de um registo seja verificado quando uma transação é realizada. Eventuais conflitos de transações têm de ser resolvidos na lógica da aplicação e não pela base de dados (Kolomicenko et al., 2013).

3.6 Comparação entre os sistemas NoSQL referidos

Nesta secção serão realizadas comparações diretas e tabeladas sobre diferentes aspetos inerentes aos sistemas *NoSQL*, salientados no ponto anterior. Esta comparação incidirá sobre três pontos-chave: *Queries*, Distribuição e Integridade. A estrutura apresentada por cada tabela, tem a sua inspiração em esquemas apresentados nos artigos (Grolinger et al., 2013), (Moniruzzaman e Hossain, 2013), (Bonnet et al., 2011), (Hecht e Jablonski, 2011) e no endereço *web* (SolidIT, 2014), um sítio dedicado à análise de caraterísticas de sistemas relacionais e *NoSQL*.

Legenda:

✘ - Não ✓- Sim

✘/✓ - Sim, mas não de uma forma direta ✓/✘ - Sim, mas de uma forma muito restrita

? – Não foi possível averiguar a resposta com um grau de certeza aceitável

3.6.1 Conceitos relacionados com Queries

Tabela 2 – Comparação entre os sistemas *NoSQL*: *Queries*

	Suporte *Map Reduce*	**REST**	**Queries "quase" *SQL***	**Outros Métodos de Acesso**	***Server side Scripts***	**Índices Secundários**
	Armazenamento do tipo Documentos					
MongoDB	✓	✓	×	Linguagem proprietária, CLI e APIs em várias linguagens	✓ *JavaScript*	✓
CouchDB	✓	✓	×	*APIs* em várias linguagens	✓ *JavaScript*	✓
RavenDB	✓	✓	×/✓ *LINQ Queries*	*API .NET*	✓	✓
	Armazenamento do tipo Família de Colunas					
Hbase	✓	✓	×/✓ Através de outros sistemas	*CLI, API Java* e *Thrift*	✓ *Java*	×
Cassandra	✓	×/✓ *APIs* de terceiros	✓ *CQL*	*CLI* e *APIs* em várias linguagens, incluindo *Thrift*	×	✓/×
HyperTable	✓	×	× Em Desenvolvimento	Suporte para *C++*, *Java*, *Perl*, *PHP*, *Python*, *Ruby* e *Thrift*	×	×
	Armazenamento do tipo Par Chave-Valor					
Riak	✓	✓	×	*CLI* e *APIs* em várias linguagens	✓ *JavaScript* e *Erlang*	✓
Redis	×	×/✓ *APIs* de terceiros	×	*CLI* e *APIs* em várias linguagens	✓ *Lua*	×
Voldemort	✓	✓	×	*API* Java e clientes *C++*, *Python* e *Ruby*	?	?
DynamoDB	✓ *Amazon Elastic Map Reduce*	✓	×	Linguagem propretária, *APIs* em várias linguagens	×	×
	Armazenamento do tipo Grafo					
Neo4J	×	✓	×	Linguagem *Cypher*, *CLI* e *APIs* em várias linguagens	✓ *Java*	✓
Titan	×/✓ através do sistema *Faunus*	×	×	*API Java* e *TinkerPop stack,* suporte para *Clojure* e *Python*	✓	✓
OrientDB	×	×	✓	Suporte para várias linguagens incluindo *TinkerPop Stack*	✓ *Java* e *JavaScript*	✓

A capacidade que um sistema possui para realizar *queries*, está entre os pontos mais importantes a analisar aquando da seleção de um sistema *NoSQL*. Os diferentes tipos de modelo de dados influenciam diretamente o modo como as *queries* (e os resultados das mesmas) são operadas. Ao mesmo tempo, diferentes sistemas fornecem diferentes *APIs* e modos de acesso aos dados. Por exemplo, um sistema que possua um armazenamento do tipo par chave-valor, não pode fornecer *queries* baseadas no conteúdo do valor, porque estes valores são opacos para o sistema (Grolinger et al., 2013). Um sistema com armazenamento de tipo documentos, já consegue realizar esta tarefa, pois o seu modelo de dados permite indexar e realizar *queries* sobre o conteúdo de documentos.

O primeiro item analisado na tabela anterior é o suporte que cada sistema apresenta para *MapReduce*. Como referido anteriormente, esta *framework* tornou-se na "ferramenta" de eleição para o processamento de dados num ambiente distribuído. Por conseguinte, a maioria dos sistemas estudados suporta este tipo de processamento. De salientar que os sistemas *NoSQL* de armazenamento do tipo grafo não suportam *queries* segundo esta *framework*, mas já o sistema *Titan* consegue responder a tais tarefas quando conjugado com o sistema *Faunus* (Aurelius, 2014b). De salientar ainda que o sistema *DynamoDB* apresenta apenas suporte para uma versão proprietária do algoritmo *MapReduce*, denominado de *Amazon Elastic MapReduce* (Amazon, 2014b). Os dois seguintes itens analisados dizem respeito aos métodos de acesso e interação com o sistema.

As *APIs* baseadas em *REST* têm vindo a ganhar tração e popularidade no mundo das aplicações *Web*, muito em parte devido à simplicidade que lhes é inerente. Desta forma, a maioria dos sistemas estudados apresenta suporte para este ponto de modo direto, ou através de sistemas de terceiros.

Os conceitos e linguagem *SQL* estão cada vez mais presentes no mundo *NoSQL*. O facto de ser uma linguagem globalmente bem conhecida e estabelecida há algumas décadas, acaba por a tornar no método de realização de *queries* preferencial para a maioria dos utilizadores. Assim sendo, os principais sistemas *NoSQL* providenciam linguagens altamente baseadas na filosofia *SQL*. Na terceira coluna da tabela anterior, no entanto, não se pretendeu aferir quais eram os sistemas que forneciam linguagens baseados na filosofia *SQL*, mas sim sistemas que possuíam uma linguagem bastante parecida com esta. Os resultados foram ligeiramente desapontantes, com apenas alguns sistemas a apresentarem resultados positivos.

Um *CLI*[9] (***C**ommand-**L**ine **I**nterface*) é a ferramenta mais simples utilizada para se interagir com vários sistemas. Por consequência, é o tipo de interface mais disponibilizado no mundo *NoSQL*, como pode ser visualizado na quarta coluna da tabela referida. Ao mesmo tempo várias outras *APIs* são fornecidas pelos vários sistemas, excluindo o sistema *RavenDB*. É ainda salientável o facto de que os sistemas *Titan* e *OrientDB* fornecem suporte para *queries* baseadas na *framework* de computação de grafos *TinkerPop* (TinkerPop, 2014).

[9] Um **I**nterface de **L**inha de **C**omandos representa um meio de interação com um computador, em que um utilizador fornece comandos a um programa, através de linhas sucessivas de texto.

Apesar de serem ligeiramente deslocados do contexto da realização de *queries*, os *server side scripts,* que neste plano tomam o sinónimo de *stored procedures,* apresentam um método capaz de reduzir a complexidade de escrita de *queries* intrínsecas, para além de possibilitar um aumento de performance. Os resultados obtidos indicam que todos os sistemas estudados de tipo de armazenamento grafo e documento, suportam plenamente este ponto. Revelando uma cada vez maior aproximação a algumas características inerentes aos tradicionais sistemas relacionais. A última coluna da tabela ilustrada, representa o suporte para a criação de índices secundários por parte de cada sistema. Os índices secundários podem revelar-se num componente essencial para a redução do tempo de execução de uma *querie*. O Estudo revela, mais uma vez que todos os sistemas de armazenamento do tipo grafo e documento suportam este ponto, e que mais de 50% dos sistemas estudados fornecem suporte para a construção de índices secundários.

3.6.2 Conceitos de Distribuição e integridade

Uma das caraterísticas mais importantes dos sistemas *NoSQL* é a sua capacidade de escalar horizontalmente, de forma eficiente, através da adição de mais servidores à lista de recursos disponíveis. Segundo (Grolinger et al., 2013) no processo de escalar horizontalmente recursos devem ser consideradas três dimensões: escalar pedidos de leitura, escalar pedidos de escrita, escalar espaço de armazenamento. As definições de particionamento, replicação, consistência e controlo de concorrência apresentadas por cada sistema têm um forte impacto na sua escalabilidade.

3.6.2.1 Particionamento

As definições de particionamento determinam o modo como os dados são distribuídos pelos vários servidores, por conseguinte, com este método é possível alcançar todas as três dimensões de "escalabilidade" referidas anteriormente. Como pode ser observado na primeira coluna da Tabela 3, a grande maioria dos sistemas *NoSQL* disponibiliza um tipo de particionamento denominado de *Sharding*.

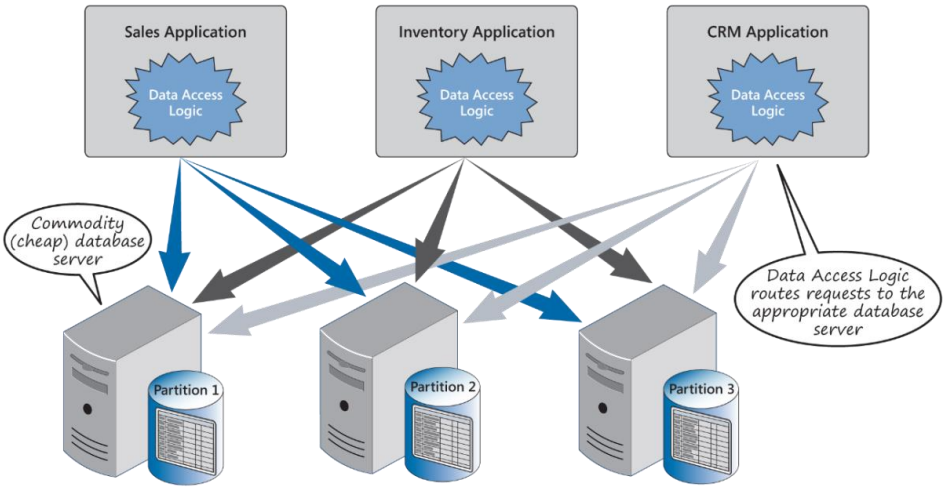

Figura 11 – Demonstração do conceito *Sharding*. Retirado de (McMurtry et al., 2013)

Sharding neste contexto torna-se num sinónimo de particionamento horizontal e pode ser traduzido como "fragmentação". Segundo (Liu et al., 2012) , *Sharding* é o processo de divisão, distribuição e armazenamento de porções de dados através de diferentes servidores. Ao alcançar-se esta distribuição, torna-se possível armazenar mais dados, sem ser necessário recorrer a *hardware* maior e mais poderoso, alcançando assim uma maior contenção de custos. De acordo com a tabela referida, os métodos de particionamento horizontal mais utilizados são *range partitioning* e *consistent hashing.*

Range partitioning é um método em que são atribuídos dados a partições localizadas em diferentes servidores, baseando-se para esse efeito em intervalos de uma chave de partição. Um servidor é responsável pelo armazenamento, bem como pelo tratamento de pedidos de escrita e leitura de um intervalo de chaves específico. Devido ao facto de chaves adjacentes normalmente estarem presentes no mesmo nó, este método torna-se especialmente eficiente quando é necessários processar *queries* baseadas em intervalos (Grolinger et al., 2013). Por outro lado, este método pode originar problemas de equilíbrio na distribuição de cargas de trabalho pelos vários servidores. Esta situação poderá levar a graves sobrecargas em alguns dos nós. Apenas três dos sistemas analisados apresentam suporte para este método de particionamento horizontal.

No método *Consistent Hashing*, o conjunto de dados é representado sob a forma de um círculo ou anel. Este círculo está dividido num número de intervalos que é igual ao número de nós disponíveis. Nesta situação cada nó encontra-se mapeado num ponto desse círculo.

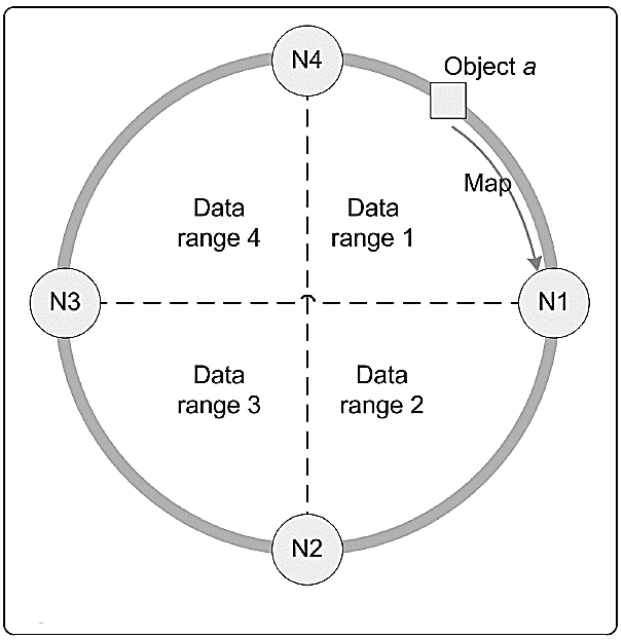

Figura 12 – Exemplo de *Consistent Hashing*. Retirado de (Grolinger et al., 2013)

Tabela 3 - Comparação entre os sistemas NoSQL: Integridade e Distribuição

	Particionamento	**Replicação**	**Consistência**	**Controlo de Concorrência**
	Armazenamento do tipo Documentos			
MongoDB	Sharding - *Range Partitioning*	Mestre-Escravo assíncrona	Configurável: Eventual consistência, Forte consistência	Bloqueio de Escritas e Leituras
CouchDB	Sharding - *Consistent Hashing*	Mestre-Escravo e Multi-Mestre	Eventual consistência	*MVCC*
RavenDB	Sharding - **?**	Multi-Mestre	*Eventual consistência*	Controlo de Concorrência otimista
	Armazenamento do tipo Família de Colunas			
Hbase	*Sharding - Range Partitioning*	Mestre-Escravo e Multi-Mestre, assíncrona	Forte consistência	*MVCC*
Cassandra	Sharding - *Range Partitioning* ou *Consistent Hashing*	Assíncrona e sem mestre	Configurável: Eventual consistência, Forte consistência	Timestamps são utilizados para determinar a atualização mais recente a uma coluna
HyperTable	*Sharding - Consistent Hashing*	Mestre-Escravo	Forte consistência	*MVCC*
	Armazenamento do tipo Par Chave-Valor			
Riak	*Sharding - Consistent Hashing*	Assíncrona e sem mestre	Eventual consistência	*MVCC*
Redis	**×** Em Desenvolvimento	Mestre-Escravo assíncrona	Configurável: Eventual consistência, Forte consistência	Controlo de Concorrência otimista ou pessimista
Voldemort	*Sharding - Consistent Hashing*	Assíncrona e sem mestre	Eventual consistência	*MVCC*
DynamoDB	*Sharding - Consistent Hashing*	Síncrona e tripartida em várias zonas numa região	Eventual consistência, Forte consistência	Controlo de Concorrência otimista ou pessimista
	Armazenamento do tipo Grafo			
Neo4J	**×**	Mestre-Escravo	Eventual	Bloqueio de Escritas
Titan	**?**	**?**	Configurável: Eventual consistência, Forte consistência	Controlo de Concorrência otimista
OrientDB	*Sharding - Consistent Hashing*	Multi-Mestre	Forte consistência	*MVCC*

Para descobrir o nó onde um objeto deve ser armazenado, o sistema passa a chave desse objeto por uma função de *hash,* e descobre a sua posição no círculo. Posteriormente, o círculo é percorrido no sentido dos ponteiros do relógio, e o objeto é atribuído ao primeiro nó encontrado, esta situação encontra-se ilustrada na Figura 12. Desse modo, cada nó fica responsável pela zona do círculo entre ele próprio e o seu antecessor. A vantagem deste método é que se torna bastante rápido calcular a localização dum objeto. Este método também se revela superior em situações de redimensionamento dinâmico. Se um nó for adicionado ou removido do círculo, apenas as regiões vizinhas dos nós serão reatribuídas a diferentes nós. Desta forma, a maioria dos registos irão permanecer inalterados. Este método apresenta uma desvantagem de performance no caso de *queries* baseadas em intervalos, visto que as chaves vizinhas estão distribuídas por nós distintos (Grolinger et al., 2013), ao contrário do que acontece no *range partitioning*. A grande maioria dos sistemas *NoSQL* analisados suporta este método de particionamento. É necessário salientar que o eficiente particionamento de dados armazenados em grafos continua a ser um problema em aberto (Kolomicenko et al., 2013). Por conseguinte o sistema *Neo4J* não apresenta nenhum tipo particionamento horizontal. Não foi possível averiguar se o sistema *Titan* apresenta suporte para particionamento horizontal. Apesar de ser um sistema de armazenamento do tipo grafo, os criadores do sistema *OrientDB* reivindicam que o seu sistema suporta particionamento horizontal, para esse efeito recorrendo ao sistema *Cassandra* (Garulli, 2014).

3.6.2.2 Replicação

Outro importante fator no processo de escalar horizontalmente pedidos de leituras e escritas são as definições de replicação do sistema em causa. Armazenar os mesmos dados em diferentes servidores, garante que pedidos de escrita/leitura possam ser distribuídos pelos mesmos. As definições de replicação também se revelam bastante importantes para garantir tolerância a falha, porque a disponibilidade dos dados em vários servidores assegura a estabilidade do sistema, mesmo no evento de falhas por parte de um ou vários servidores. Existem, como observado na tabela comparativa anterior, duas abordagens distintas no que diz respeito às definições de replicação: Mestre-Escravo ou Multi-Mestre.

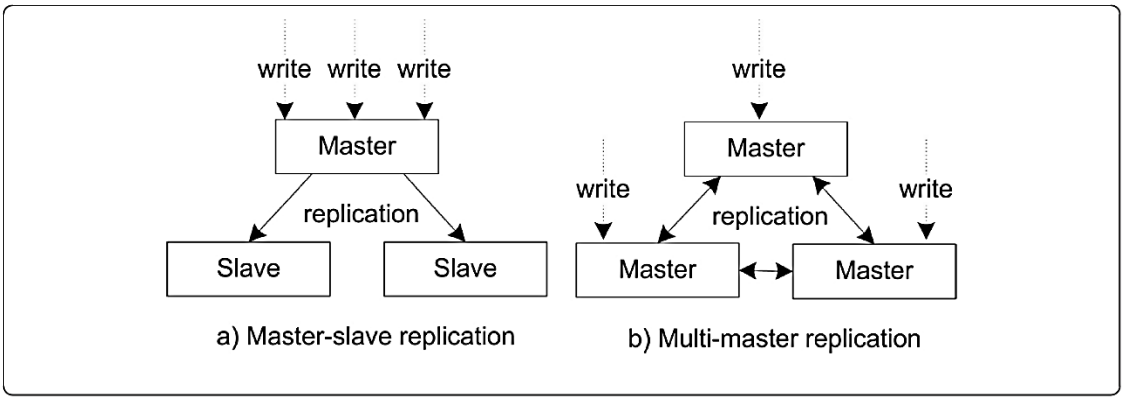

Figura 13 – Modelos de replicação. Retirado de (Grolinger et al., 2013)

No modelo de replicação Mestre-Escravo, um nó é definido como mestre, sendo que apenas este responde a pedidos de escrita. As mudanças realizadas sobre os dados são posteriormente propagadas pelos outros nós. No modelo de replicação Multi-Mestre (ou mestre-mestre), existem vários nós a realizar o processamento de pedidos de escrita, as alterações são posteriormente propagadas pelos outros nós. No caso da replicação Mestre-escravo, as propagações acontecem sempre no sentido dos escravos, já no modelo multi-mestre, as propagações podem acontecer em várias direções. Outro modelo de replicação encontrado na tabela comparativa, é o modelo "sem mestre". Este modelo é bastante similar ao modelo multi-mestre, assim, vários nós aceitam pedidos de processamento de escrita de dados, mas neste modelo todos os nós estão no mesmo nível hierárquico, não existindo nenhum mestre.

Uma caraterística do processo de replicação que pode influenciar drasticamente a velocidade de processamento e transferência dos sistemas *NoSQL* é o método de propagação de operações de escrita pelos nós. Assim, deve ser fortemente avaliado qual o processo de sincronização de réplicas a ser utilizado em cada sistema. Estes métodos podem ser: síncronos ou assíncronos. Na replicação síncrona, as alterações aos dados são propagadas pelos restantes nós antes de se comunicar ao cliente o sucesso da operação de escrita. Este método introduz um elevado valor de latência no processamento de pedidos de escrita, pois operações de escrita só são consideradas completas quando as alterações são propagadas. Deste modo, este método acaba por ser muito pouco utilizado no contexto *NoSQL* (Grolinger et al., 2013). O método de replicação assíncrono, por sua vez, apenas propaga as alterações aos outros nós, após ter confirmado com o cliente o sucesso da operação de escrita. Este garante uma maior performance, mas ao mesmo tempo, pode gerar uma grande inconsistência nos dados presentes em diferentes servidores.

3.6.2.3 Consistência

Este ponto já foi largamente abrangido neste dissertação, mais especificamente nos capítulos destinados ao estudo das propriedades *ACID*, teorema *CAP* e das propriedades *BASE*. Como é observável na Tabela 3, a maioria dos sistemas estudados rege-se pela filosofia da eventual consistência, uma caraterística adquirida das propriedades *BASE*, inerente a vários sistemas *NoSQL*. Para além da eventual consistência, muitos destes sistemas suportam simultaneamente o modelo de forte consistência, deixando para o utilizador a possibilidade de configurar qual o nível de consistência pretendido. O modelo de forte consistência assegura que depois dos dados terem sido escritos com sucesso, todos os pedidos de leitura seguintes visualizam essa mesma versão dos dados, independente do nó do sistema responsável por responder ao pedido. O modelo de replicação síncrona normalmente assegura uma forte consistência, no entanto e como referido, o mesmo introduz um elevado custo em termos de latência que normalmente não pode ser suportado pelos sistemas de várias organizações. Dos sistemas estudados, apenas o *HBase* fornece ao mesmo tempo uma replicação configurável e forte consistência. Este acaba por ser um fator interessante e ligeiramente inesperado, dadas as caraterísticas dos sistemas *NoSQL*.

3.6.2.4 Controlo de Concorrência

Os sistemas *NoSQL*, foram maioritariamente desenhados para operar em ambientes com vários utilizadores e um elevado número de pedidos de escrita/ leitura simultâneos. Assim, as técnicas e métodos empregues para efetivamente controlar o simultâneo acesso aos mesmos dados toma extrema importância no contexto *NoSQL*.

Os principais esquemas de controlo de concorrência utilizado pelos sistemas analisados podem ser categorizados como otimistas ou pessimistas. No controlo de concorrência pessimista é considerado que dois ou mais clientes tentarão atualizar o mesmo registo simultaneamente. De modo a evitar esta situação, o sistema bloqueia o registo em causa, para assim garantir exclusividade de acesso a uma só operação. Outros acessos aos mesmos registos só serão garantidos após o pedido que chegou primeiro terminar a sua tarefa. O sistema *MongoDB* implementa um controlo de concorrência pessimista de dois estados. Um deles permite que vários processos de leitura acedam a um registo, ou que apenas um processo de escrita aceda a esse registo. Para alcançar esse efeito o sistema vai bloqueando pedidos de escrita ou leitura conforme o tempo de chegada e a prioridade dos mesmos. Técnicas de bloqueio de registos pessimistas podem levar à degeneração da performance, especialmente do caso de sistemas ricos em processos de escrita.

No controlo de concorrência otimista, a filosofia passa por assumir que conflitos no acesso aos dados são possíveis, mas raros. No final da operação o sistema verifica se os mesmos dados foram modificados simultaneamente. Se uma situação conflituosa for detetada, vários métodos poderão ser aplicadas para resolver a situação, como falhar a operação ou repetir uma das operações. Neste controlo de concorrência o sistema não bloqueia acessos aos dados, limitando-se a gerir posteriormente os conflitos encontrados. Vários dos sistemas listados implementam um controlo de concorrência otimista com *MVCC*, a sigla para ***M**ulti-**V**ersion **C**oncurrency **c**ontrol* (controlo de concorrência multi-versão). Neste tipo de controlo de concorrência, quando um nó precisa de atualizar um registo, o sistema não modifica o registo selecionado, mas sim adiciona um registo novo e marca o antigo como obsoleto. Com esta abordagem, os processos de escrita têm sempre a visão do registo, como ele se encontrava no início do processo, mesmo que este tenha sido modificado ou eliminado por outro processo a decorrer simultaneamente (Grolinger et al., 2013).

3.7 Movimento *NewSQL*

A origem da palavra *NewSQL* é atribuída a um relatório elaborado por *Matt Aslett* (Aslett, 2011), onde discutia o aparecimento de novos sistemas de gestão de bases de dados, que estavam a desafiar sistemas fortemente estabelecidas neste sector de mercado. O nome surge na sequência do jogo de palavras apresentado pelo movimento *NoSQL,* e a sua interpretação literal não representa a verdadeira essência deste movimento. *NewSQL* não é uma nova iteração da amplamente adotada linguagem de quarta geração, nem representa uma nova variante do modelo relacional. *NewSQL* representa um novo conjunto de sistemas emergentes, que se propõem disponibilizar a performance e a escalabilidade dos sistemas *NoSQL*, ao mesmo tempo

preservando a linguagem SQL (Doshi et al., 2013) e as propriedades *ACID*, inerentes aos tradicionais sistemas de gestão de bases de dados relacionais (Stonebraker, 2012). Na era *Big Data* o paradigma do armazenamento de dados sofreu uma grande mutação. Uma primeira análise entregava todas as responsabilidades de armazenamento nesta nova era aos sistemas *NoSQL.* No entanto, essa análise revelou-se precoce. Em 2012, a empresa que introduziu as fundações que viriam a originar o movimento *NoSQL*, voltava a estar envolvida na introdução de um novo paradigma de armazenamento. Primeiro com a introdução do *Spanner* (Corbett et al., 2012), uma base de dados distribuída e relacional, a *Google* demonstrou que o *SQL* e a filosofia do modelo relacional ainda eram necessários para gerir e armazenar dados na era *Big Data*. De seguida, a *Google* introduziu um outro sistema *NewSQL*, o *Google F1* (Shute et al., 2012) (Vingralek et al., 2013), uma base de dados *NewSQL* distribuída e escalável, que tem por base o *Google Spanner* (Clark, 2013). Esta base de dados é a responsável pelo armazenamento daquela que é, possivelmente, a plataforma mais rentável e importante do universo *Google*, a plataforma *AdWords* [10] . Segundo (Stonebraker, 2012), os sistemas *NewSQL* devem ser considerados como uma alternativa aos sistemas *NoSQL* e aos tradicionais sistemas relacionais, quando se considera a criação de novas aplicações que requeiram o processamento de transações em tempo real. Segundo a mesma fonte, num futuro próximo devem surgir, várias novas soluções *NewSQL*, apresentando um leque variado de arquiteturas e implementações.

As soluções *NewSQL* possuem 5 características técnicas chave (Venkatesh, 2012):

- A linguagem *SQL* representa o principal meio de interação entre as aplicações e a base de dados.
- Transações regidas pelas propriedades *ACID*.
- Um mecanismo de controlo de concorrências que não recorre a bloqueio de recursos para que, assim, leituras de dados em tempo real não entrem em conflito com a escrita de dados, assim evitando falhas e interrupções no sistema.
- Uma arquitetura que proporciona uma performance por nó, muita mais elevada que aquela disponibilizada pelos sistemas de gestão de bases de dados relacionais tradicionais.
- Uma arquitetura altamente escalável, capaz de correr sobre um elevado número de nós, sem sofrer problemas de performance.

Segundo (Grolinger et al., 2013) estas são as principais soluções *NewSQL* disponíveis:

3.7.1 VoltDB (VoltDB, 2014)

O *VoltBD* é um sistema de gestão de bases de dados relacional "em memória", desenvolvido com o intuito de ser extremamente rápido (Mao et al., 2012). Para alcançar tal efeito este sistema apresenta na sua arquitetura elementos que lhe permitem apresentar uma elevada

[10] É o serviço de publicidade *online* utilizado pela *Google* no seu motor de pesquisa. Em 2013 a receita da *Google* foi de 33,3 milhares de milhão de dólares, 97% desse receita teve origem em serviços de publicidade online.

taxa de transferência e tolerância a falha quando deparado com as elevadas cargas de trabalho, inerentes às operações realizadas sobre dados transacionais (Minhas et al., 2012). Segundo a mesma fonte, esses elementos são:

- Todos os dados são armazenados na memória principal, desse modo evitando operações de disco rígido, por natureza mais lentas que operações decorridas ao nível da memória *RAM*.
- Todas as operações a realizar na base de dados, estão pré definidas num conjunto de *stored procedures* que são executados no servidor em causa, transações *ad-hoc* não são permitidas.
- Transações são executadas em série dentro de cada partição da base de dados. Sem a existência de várias transações a ocorrerem simultaneamente na mesma partição, elimina-se a necessidade de existência de um mecanismo de controlo de concorrência.
- Partições são replicadas para se conseguir alcançar durabilidade e tolerância a falha.
- A base de dados de um sistema *VoltDB* é composta por um número de partições distribuídas por vários servidores (Buckle, 2012).
- As *stored procedures* são escritas, recorrendo à linguagem *Java* (Buckle, 2012).

Apesar de o *VoltDB* possuir uma versão empresarial (paga) e uma versão da comunidade (gratuita e de fonte aberta), este sistema acaba por ser considerado como uma representação comercial dos princípios introduzidos pelo projeto *H-Store* (Kallman et al., 2008). Segundo (Weisberg e Stonebraker, 2013) o *H-Store* representa um protótipo académico de pesquisa que precedeu à criação do sistema *VoltDB*. Os dois sistemas partilham uma pessoa chave: *Michael Stonebraker*, responsável por fundar alguns projetos extremamente importantes no contexto dos sistemas de gestão de bases de dados, como os projetos *Ingres* e *PostgreSQL* (MIT-CSAIL, 2008).

3.7.2 ClustrixDB (Clustrix, 2014)

O sistema *ClustrixDB*, representa o culminar de uma oferta *NewSQL* que tem vindo a ser aprimorada desde 2006 (Higginbotham, 2010). Este sistema disponibilizado pela empresa *Clustrix*, sediada em São Francisco, começou a ser usufruído por organizações desde 2008 (DBMS2, 2013). Todavia, só desde meados de 2013 apresenta o formato e nome atuais, anteriormente este sistema tomava a designação de *Clustrix-Sierra* (Clustrix, 2012). Devido ao facto de ter as suas origens sediadas antes do fim da primeira década do século XXI e antes mesmo de a palavra *NewSQL* ter sido utilizada pela primeira vez, este sistema acaba por ser considerado como bastante maduro quando enquadrado com os seus concorrentes *NewSQL*. O *ClustrixDB* possui três versões. Uma versão standard (paga), uma versão da comunidade (livre, para utilizações até 12 nós), uma versão que recorre apenas à computação em nuvem (paga, disponível na loja de serviços da *Amazon Web Services*).

3.7.3 NuoDB (NuoDB, 2014)

É um sistema *NewSQL* disponibilizado pela organização com o mesmo nome, sediada em *Cambridge* no *Massachusetts*. Este sistema foi sendo disponibilizado como versão de teste (para elementos selecionados) desde 2010 e em janeiro de 2013 alcançou a primeira versão final e a consequente disponibilização para o público (Rouse, 2014). Um ano mais tarde foi lançada a versão *2.0* deste sistema. Segundo os seus criadores, este sistema é "simplesmente um *software*", com isso querendo dizer que este sistema não requer uma arquitetura, configuração ou sistema operativo específico. É capaz de correr tanto num conjunto de servidores, como completamente na nuvem ou num simples computador portátil doméstico (Proctor, 2012). São suportados os vários sistemas baseados em *Unix* e *Windows*. Esta situação torna fácil testar e desenvolver aplicações baseadas no sistema prontamente, deixando para mais tarde as questões relacionadas com a implementação específica do sistema no seio da organização (Proctor, 2012). Este sistema possui na sua arquitetura um conceito bastante interessante: tudo é um átomo. Apesar de ser um sistema relacional que interage com o utilizador através de *SQL*, por detrás dessa camada, toda a lógica do sistema opera sobre objetos que tomam o nome de átomos (Proctor, 2012). Átomos são objetos auto coordenados que representam um tipo específico de informação; desde dados, índices, esquemas ou os meta dados internos, todos são guardados como átomos (Proctor, 2012). Os átomos representam muito mais que uma maneira de organizar dados, os mesmos simplificam largamente os processos internos de comunicação e operações de cache, porque não é necessário pensar ou materializar uma estrutura *SQL* específica (Proctor, 2012). Tudo está contido na mesma estrutura genérica e identificado de uma maneira consistente. Este sistema possui 5 versões: versão de programador e versão livre (ambas livres de custos, mas com limites na versão livre), versão *Pro* e versão empresarial (ambas pagas) e a versão que recorre apenas à computação em nuvem (paga, disponível na loja de serviços da *Amazon Web Services*).

3.8 Conclusões

Este capítulo apresenta-se como o resultado de uma investigação profunda, que incidiu unicamente sobre o conceito *NoSQL*. Investigaram-se os vários tipos de armazenamento disponíveis, bem como os principais sistemas englobados em cada um deles.

O capítulo terminou com uma incursão por um novo conceito que não se conhecia antes do início da investigação aqui realizada. As bases de dados *NewSQL,* apresentam-se como uma alternativa aos sistemas relacionais e também aos sistemas *NoSQL*. Alguns estudos foram elaborados com o sistema *NuoDB,* tendo em vista a sua integração nos testes de performance a realizar num ponto posterior deste projeto. Infelizmente, os resultados iniciais obtidos com estes sistemas foram bastante negativos, pelo que a sua utilização teve de ser descartada.

Antes, no entanto, de se terminar o capítulo, foi realizada uma comparação exaustiva sobre as caraterísticas tecnológicas dos diferentes sistemas salientados.

4 Armazéns de Dados e *NoSQL*?

4.1 Introdução

Como referido no capítulo introdutório deste documento, um dos principais objetivos a alcançar com esta dissertação é o estudo da viabilidade da utilização de bases de dados NoSQL em armazéns de dados. Para se aferir se é realmente útil substituir as bases de dados relacionais, consideradas como *standard* neste contexto, é necessário criar primeiramente um armazém de dados para servir de comparação. Não se pretende que esta comparação se limite apenas à performance de consultas. Fatores como o esquema das bases de dados, os processos de migração de informação e a linguagem utilizada para realizar consultas, têm um grande impacto na utilização de um sistema.

Este capítulo representa o culminar de vários estudos que resultaram na criação da estrutura de um armazém de dados relacional e na seleção dos sistemas a utilizar. Nesta secção serão também definidos quais os testes de performance a realizar. Antes no entanto de se chegar a este ponto mais prático, será realizada uma ponte entre os temas armazéns de dados e NoSQL através da referência a conceitos inerentes ao desenho de esquemas em bases de dados deste género.

4.2 Sistemas e critérios de seleção

Foram vários os sistemas estudados ao longo dos primeiros estágios da elaboração desta dissertação. Infelizmente, a profundidade que se pretende alcançar com este estudo conjugada com o fator tempo levam a que seja necessário aplicar restrições ao número de sistemas a utilizar. Deste modo, alguns estudos laterais tiveram de ser realizados, e alguns objetivos extra

foram definidos de modo a limitar mais facilmente a escolha dos sistemas a testar neste caso de estudo.

Um fator importante a não descurar no processo de seleção dos sistemas, é o sistema operativo no qual serão implementados não só os sistemas *per se*, mas também todos os processos de integração, validação e teste. Devido ao facto de os sistemas *NoSQL* terem como maior área de utilização a *web*, a maioria dos sistemas têm por base o ambiente *Linux*. Uma dissertação representa uma oportunidade única para aprender e melhorar conhecimentos. Assim sendo, um dos objetivos (pessoais) a alcançar neste projeto visava melhorar e expandir os conhecimentos na linguagem *C#*, visto ser uma linguagem que foi invariavelmente preterida na realização de vários projetos. Como a linguagem *C#* é proprietária da *Microsoft*, a mesma só pode ser executada e programada sobre uma plataforma *Windows*. Deste modo acabou por surgir um dos objetivos secundários desta dissertação: investigar a possibilidade de implementação dos sistemas *NoSQL* selecionados e as respetivas consultas num ambiente *Windows*.

O sistema que irá servir de base a todas as comparações será um sistema de gestão de bases de dados relacional "tradicional". Assim revela-se essencial escolher apropriadamente o sistema em causa. Vários sistemas foram considerados, mas apenas 3 alcançaram o processo de comparação final: *Microsoft SQL Server* (Microsoft, 2014e), *Oracle Database* (Oracle, 2014), *PostgreSQL* (PostgreSQL, 2014). Apenas este último possui uma utilização totalmente livre de custos, mas como aluno do Instituto Superior de Engenharia do Porto, o acesso aos outros dois sistemas está igualmente garantido sem custos adicionais. Todos os três sistemas possuem versões destinadas a correr na plataforma Windows. A decisão final recaiu sobre o sistema *SQL Server* devido, em grande parte, ao facto de ser o sistema em que se possui maior experiência, e também por na sua instalação incluir aplicações destinadas a apoiar o processo de integração e criação de armazéns de dados.

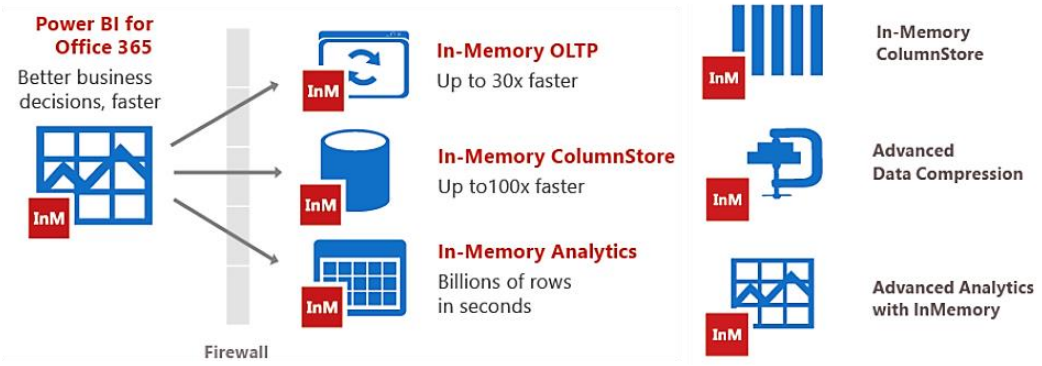

Figura 14 – Características do sistema *SQL Server 2014*. Adaptado de (Microsoft, 2014e)

Como referido, o número de sistemas *NoSQL* a utilizar também teve de ser limitado. Estabeleceu-se que seriam utilizados dois sistemas *NoSQL* distintos, e o nível de popularidade foi o critério de seleção estabelecido como fator de seleção. Infelizmente o critério escolhido é bastante obtuso e difícil de quantificar. Existe, no entanto, um estudo periódico levado a cabo pela organização *451 Research (Group, 2014)*, que investiga quais os sistemas *NoSQL* mais

referidos pelos utilizadores da rede social *LinkedIn* como sendo uma "competência". Foi neste estudo que se baseou a seleção dos sistemas a utilizar.

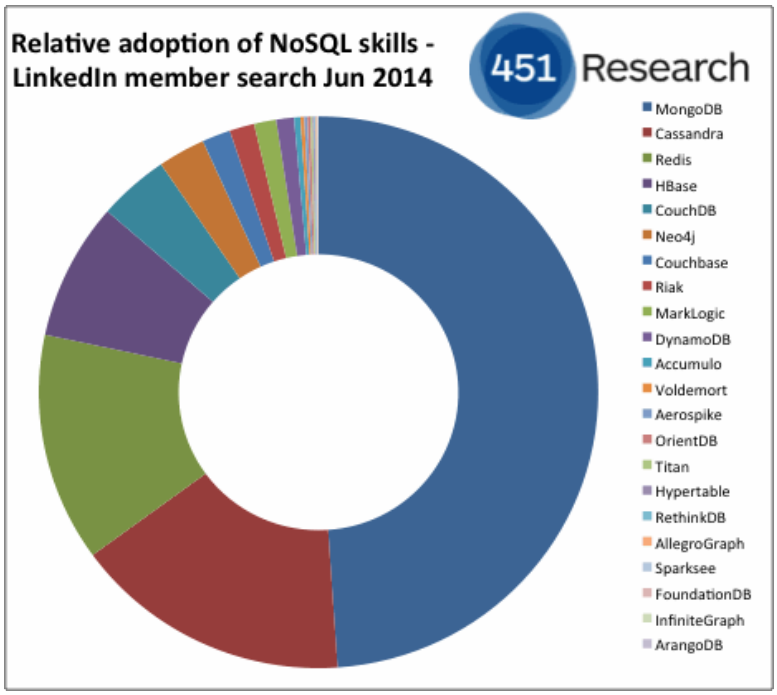

Figura 15 - Adoção dos sistemas NoSQL, segundo informações dispostas no linkedin (Aslett, 2014)

O primeiro ponto a concluir da análise das duas imagens que circundam este parágrafo é o facto de o sistema *MongoDB* ser o rei indiscutível no que à popularidade diz respeito. Também é possível observar que no espaço de cerca de dois anos, a popularidade dos sistemas *NoSQL* aumentou exponencialmente. Se o foco for apenas no sistema *MongoDB*, a escala de evolução na adoção é ainda maior. Em menos de dois anos o *MongoDB* aumentou as suas referências em cerca de oitenta milhares. A luta pelo segundo lugar é bastante mais próxima, mas o sistema Cassandra assume a liderança sobre o sistema Redis. Assim, os sistemas *NoSQL* selecionados são o sistema *MongoDB* e o sistema *Cassandra*.

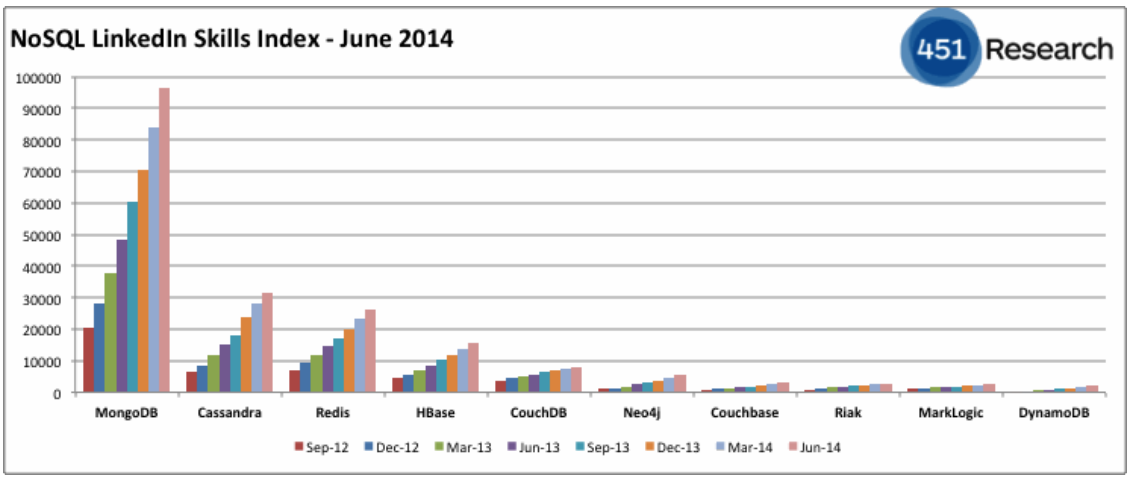

Figura 16 – Histórico da adoção de sistemas *NoSQL*, na rede social *LinkedIn* (Aslett, 2014)

4.3 Armazém de dados

4.3.1 Seleção da fonte de dados

O primeiro passo a realizar após a seleção dos sistemas a utilizar é a criação de um armazém de dados que irá servir de base a todas as comparações. Para a construção do mesmo foi necessário perscrutar várias fontes de dados.

Para alcançar os objetivos estabelecidos era necessário construir um sistema capaz de ser desafiado pela complexidade e tamanho dos dados armazenados. Várias fontes de dados foram analisadas, desde informações governamentais, a históricos de temperaturas e precipitação, mas infelizmente os dados adquiridos eram de âmbitos demasiados díspares, sendo assim bastante difícil e trabalhoso moldar esses dados de forma a emularem um armazém de dados empresarial. Após algum tempo de estudo, tornou-se claro que seria necessário recorrer a dados provenientes de uma base de dados operacional integrada numa organização. Nesta situação, as relações entre os dados são facilmente entendidas e moldadas, porque o âmbito de qualquer organização inclui sempre aspetos já amplamente trabalhados, como compras, vendas, produtos, fornecedores… Infelizmente não existem muitas organizações a disponibilizarem os seus dados operacionais para o público. No entanto, vários fornecedores de sistemas de bases de dados fornecem uma base de dados de amostra, para que se possam realizar testes e tutoriais sobre os seus sistemas. Foram estudadas as bases de dados de amostra dos principais sistemas relacionais e a base de dados de dados *AdventureWorks* (Microsoft, 2014a), apresenta-se como a mais desenvolvida e complexa de todas.

A base de dados *AdventureWorks*, foi inicialmente produzida para acompanhar a versão 2008 do sistema *SQL Server*. Esta base de dados serve de suporte a todos os processos de transação de dados necessários para suportar o negócio da empresa (fictícia) de manufatura de bicicletas "*Adventure Works Cycles*" e é bastante complexa: nela estão contidas vários cenários destinados a suportar as diferentes áreas da organização. Apesar de esta fonte de dados servir todas as necessidades em termos de estrutura, a mesma não fornece o volume de dados necessário para satisfazer os objetivos pretendidos nesta dissertação. Vários estudos foram levados a cabo de modo a ampliar o volume de dados da base de dados em causa. Este estudo levou a dois sítios web que disponibilizam excertos de código *SQL Server*. Este código centra-se essencialmente em duas tabelas: produtos e linhas de venda. O que se verificou como uma situação ótima, visto que a tabela de factos do armazém de dados irá assentar maioritariamente sobre as linhas referentes a vendas de produtos. Estes dois excertos de código foram obtidos de (Machanic, 2011) e (Kehayias, 2012).

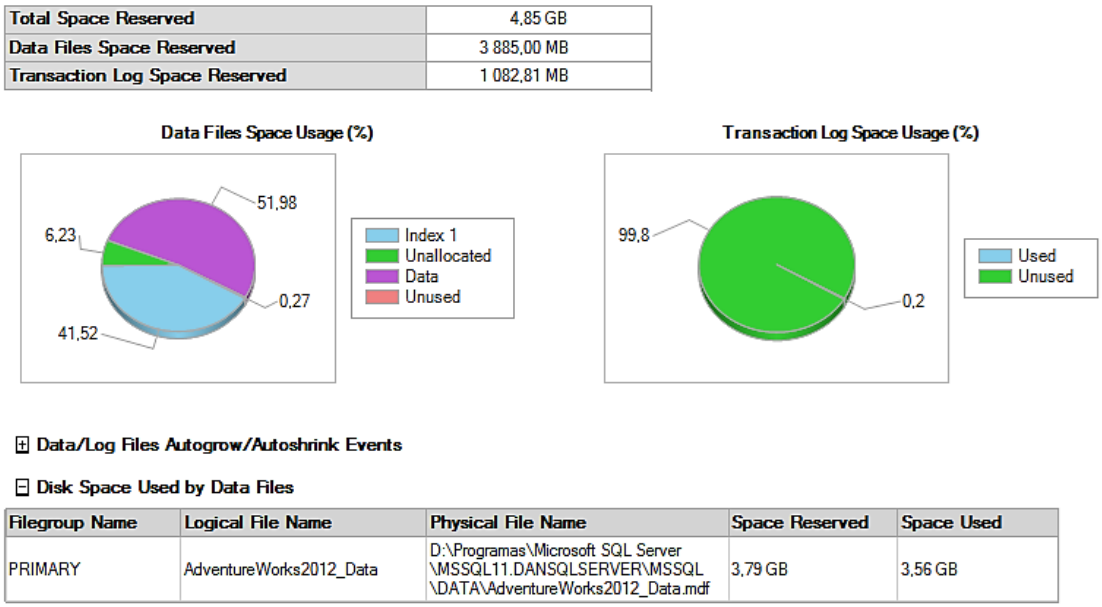

Figura 17 – Total de espaço ocupado em disco pela base de dados *AdventureWorks* modificada

Na imagem anterior está ilustrado o espaço em disco ocupado pela base de dados *AdventureWorks*, após terem sido executados os excertos de código referidos. Apesar de o espaço ocupado parecer pouco para colocar em causa a performance de um armazém de dados empresarial, a mesma já representa um desafio bastante complexo para a capacidade de processamento da máquina utilizada. Vários desafios foram surgindo ao longo do processo de implementação desta dissertação, devido à falta de capacidade da máquina em causa.

4.3.2 Esquema e desenho do armazém de dados

O processo de desenho do armazém de dados teve todas as suas bases nas metodologias de modelação multidimensional apresentadas por (Kimball e Ross, 2013) e (Imhoff et al., 2003), para além de todo o conhecimento adquirido nas aulas da unidade curricular de *Armazenamento e Processamento Analítico de dados* do curso referido na folha de rosto deste documento. Devido a questões de complexidade estrutural e de tempo de implementação, foi estabelecido que a estrutura do armazém de dados adotaria um esquema em estrela "puro", sem ramificações em floco de neve. Também foi estabelecido (pelo mesmo motivo) que o número de dimensões seria limitado a seis dimensões, não incluindo a dimensão data. Ao mesmo tempo tentou-se criar dimensões com um número de colunas ilustrativo de um bom desafio para um armazém de dados tradicional. Assim, o armazém de dados criado, possui 180 colunas, se somados todos os atributos das 7 dimensões.

Este elevado número de atributos veio a tornar-se um problema mais tarde ao nível da migração dos dados do armazém de dados relacional para as bases de dados *NoSQL*. Um outro fator a causar dificuldade nos estágios mais avançados do processo de migração de dados, foi o número de linhas presentes na tabela de factos: cerca de 5 milhões. Este número encaixa em grande parte nos objetivos de volume de dados pretendidos. Apesar de (provavelmente) não

apresentar qualquer problema para a performance de um sistema empresarial, para uma máquina e disco rígido com bastante utilização durante os últimos 6 anos, revelou-se um volume bastante elevado, tanto no processo de migração, como na realização dos testes de performance.

Na Figura 18 pode ser visualizada a estrutura do armazém de dados, bem como as medidas presentes na tabela de factos. Devido ao elevado número de atributos, o diagrama completo teve de ser remetido para o Anexo C deste documento.

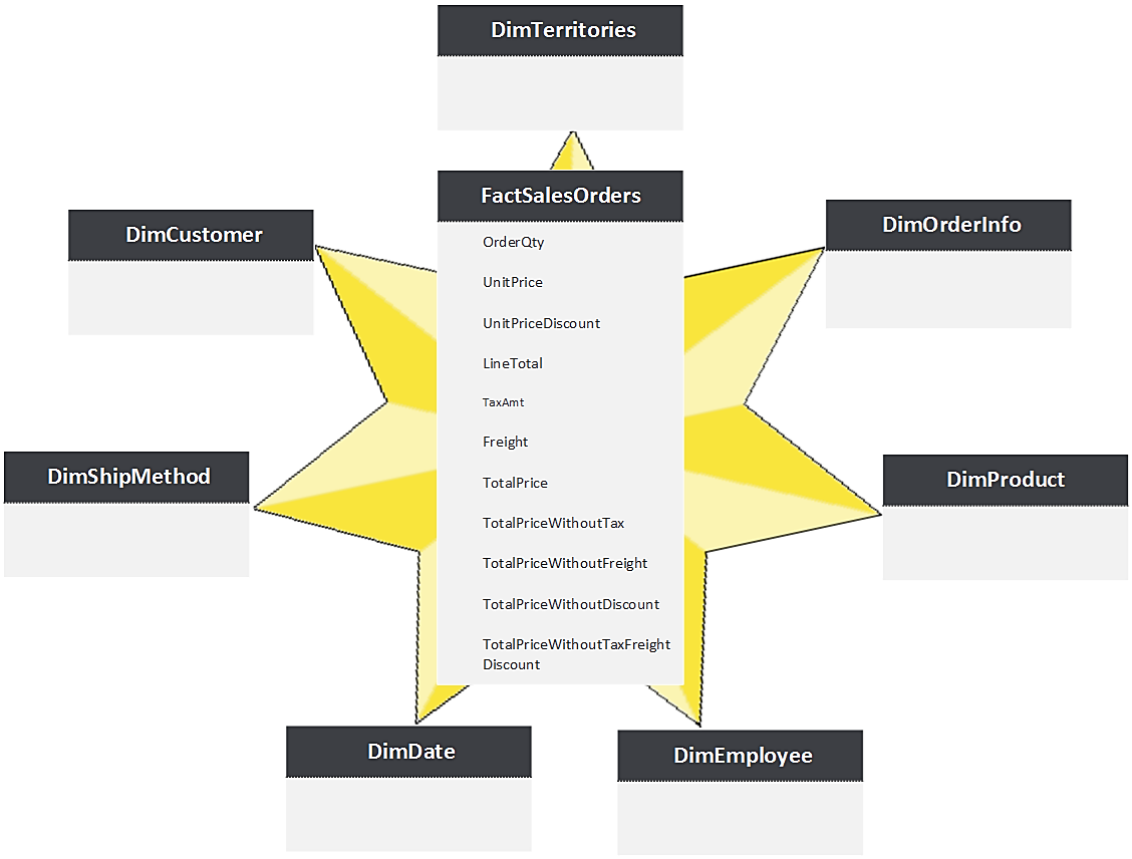

Figura 18 – Esquema em estrela ilustrativo da estrutura do armazém de dados criado

4.4 A importância das consultas para o esquema das bases de dados *NoSQL*

Em contraste com os sistemas relacionais, os sistemas NoSQL dispensam parte dos processos preparatórios que culminam no desenho de um esquema de base de dados. Em alguns dos sistemas é possível importar dados sem definir esquema, tipos de dados para colunas, chaves primárias ou relações entre tabelas (Lee et al., 2013). Assim, é possível encurtar o esforço e o tempo despendidos na implementação de bases de dados NoSQL, comparativamente com bases de dados relacionais.

Apesar de, como referido, uma das caraterísticas mais interessantes dos sistemas *NoSQL* ser a sua implementação "*schema-free*", na grande maioria das situações práticas (incluindo a situação exposta nesta dissertação) o conceito "sem-esquema" acaba por tomar na realidade o valor de "parcialmente sem-esquema". O que isto significa é que, num grande número de situações, o responsável pelo sistema ou programador da aplicação terá de criar e gerir algum tipo de esquema e restrições, mesmo que estas não sejam impostas pela base de dados. Estas restrições podem ser meramente implementadas ao nível do código/lógica da aplicação para, desse modo, conseguir alcançar objetivos específicos. Normalmente, estes incluem garantir uma boa performance de leitura e/ou escrita, pouca latência no acesso aos dados e responder de forma assertiva a determinadas consultas. O que esta situação traduz para a realidade prática é que, apesar de ser possível ignorar parte das preocupações de desenho da base de dados e do respetivo esquema num primeiro plano, de modo a alcançar uma boa performance e fiabilidade de resultados, o processo de desenho de um esquema continua a ser uma tarefa crítica e extremamente importante no processo de implementação. Esta conclusão não foi abduzida de nenhum artigo ou livro, mas sim da própria experiência adquirida na utilização dos dois sistemas *NoSQL* selecionados e que representa uma retrospetiva de conhecimento adquirido e exposto nos capítulos 5.2 e 6.2 deste documento.

Também por contraposto aos sistemas relacionais, o desenho do modelo de dados numa base de dados NoSQL está mais focado nas *queries* que irão ser executadas ao longo do tempo, do que nas entidades e relacionamentos (Vaish, 2013). Assim, revela-se bastante importante conhecer as *queries* e/ou o âmbito específico das mesmas, antes de se iniciar a modelação do sistema. Não é, todavia, necessário visionar um futuro distante e imaginar as *queries* que não se conhece. O facto de a grande maioria das bases de dados NoSQL serem "livres de esquema" permite que seja bastante simples adaptar um modelo de dados a novas variáveis e necessidades que tenham surgido. Mas, antes de essas alterações serem levadas a cabo, as mesmas necessitam de ser bem calculadas e estudadas, de modo a que tal não influencie negativamente a performance do sistema ou *queries* definidas anteriormente. De uma forma pragmática é possível afirmar que, nos sistemas relacionais são criadas *queries* em conformidade com o esquema e modelo de dados definido e, no caso dos sistemas *NoSQL*, é desenhado o esquema e modelo de dados em conformidade com as *queries* definidas.

De modo a ser possível desenhar de forma correta o modelo de dados nos dois sistemas *NoSQL* selecionados, na secção seguinte serão definidas as consultas que irão interagir com os sistemas aquando da realização dos testes de performance. Como todos os sistemas selecionados apresentam uma estrutura de armazenamento diferente, os esquemas e modelos de dados também serão respetivamente diferentes. É possível mapear *queries SQL* para as linguagens dos outros sistemas, mas como os esquemas de base de dados são diferentes, isso não seria uma solução correta para o problema proposto. Assim, as consultas serão escritas em linguagem natural e posteriormente serão adaptadas à linguagem e esquema de cada sistema. Esta dissertação tem apenas por objetivo comparar a performance dos sistemas no que diz respeito a consultas, não tendo sido assim consideradas *queries* relativas à inserção, atualização ou eliminação de dados, sendo que, no caso de armazéns de dados, as eliminações de informação até são acontecimentos bastante raros.

4.4.1 Consultas a realizar:

As consultas elaboradas têm por inspiração as *queries SQL* definidas no artigo intitulado de "*Star Schema Benchmark*" (O'Neil et al., 2009). Este artigo apresenta um conjunto de medidas que visam testar e medir a performance de sistemas de gestão de bases de dados, que servem de suporte a armazéns de dados "tradicionais" e é altamente baseado no "sistema" de testes de performance *TPC-H*[11]. As consultas criadas esperam constituir um desafio crescente para os sistemas selecionados, ao mesmo tempo representando a estrutura típica de análises multidimensionais realizadas sobre armazéns de dados.

4.4.1.1 Linguagem Natural

- **Consulta A:** Valor total dos descontos concedidos a encomendas de mais de 1, mas menos de 11 produtos. Apenas para vendas realizadas no ano de 2008.

- **Consulta B:** O valor total dos portes de envio, por cada ano de operação, e por cada produto da categoria "acessórios", cujo envio foi tratado pela empresa "*Cargo Transport 5". Os* resultados devem ser ordenados de forma ascendente pelo ano em que foi efetuado a encomenda e pelo nome do produto.

- **Consulta C:** O valor médio do preço de todas as vendas realizadas por um funcionário do sexo feminino para cada categoria de produto de cor vermelha, e para cada um dos territórios. Devem ser excluídas vendas cujos clientes sejam moradores dos Estados Unidos da América. Apenas devem ser consideradas vendas realizadas entre 1 de Junho de 2010 e 1 de Junho de 2012.

- **Consulta D:** O valor médio da quota de mercado de cada funcionário; o valor correspondente à soma do preço unitário, retirando o desconto, de cada produto presente numa venda; e o valor médio cobrado por cada fornecedor para cada um dos produtos vendidos, para cada ano de operação, e para cada um dos territórios, excluindo a Austrália. Só devem ser consideradas vendas efetuadas entre 1 de Março de 2005 e 1 de Março de 2014 e cuja entrega não esteve a cargo da empresa "*Cargo Transport 5*".

[11] O ***T**ransactional **P**rocessing Performance **C**ouncil* (***TPC***) é uma organização sem fins lucrativos criada em 1988, que tem por objetivo definir testes de performance, que atuam sobre sistemas bases de dados. O ***TPC-H*** é um teste de performance, que incide sobre sistemas de apoio à decisão responsáveis por analisar grandes volumes de dados e executar *queries* com elevado grau de complexidade.

4.4.1.2 Linguagem SQL - SQL Server 2014

- **Consulta A:**

```
SELECT SUM(UnitPriceDiscount*OrderQuantity) AS Total
FROM FactSalesOrders fact, DimDate data
WHERE fact.OrderDateKey=data.DateKey AND
      data.CalendarYear=2008 AND
      fact.OrderQuantity BETwEEN 2 AND 10
```

Código 1 – Consulta A representada na linguagem SQL

- **Consulta B:**

```
SELECT prod.Name, data.CalendarYear ,SUM(fact.Freight) AS Total
FROM   FactSalesOrders fact, DimProduct prod, DimDate data,
       DimShipMethod ship
WHERE  fact.OrderDateKey=data.DateKey AND
       fact.ProductKey=prod.ProductKey AND
       fact.ShipMethodKey=ship.ShipMethodKey AND
       prod.ProductCategoryName='Accessories' AND
       ship.Name='CARGO TRANSPORT 5'
GROUP BY data.CalendarYear, prod.Name
```

Código 2 – Consulta B representada na linguagem SQL

- **Consulta C:**

```
SELECT ter.Name, prod.ProductCategoryName, AVG(fact.TotalPrice) AS média
FROM FactSalesOrders fact, DimProduct prod, DimDate data, DimCustomer cli,
     DimTerritories ter, DimEmployee emp
WHERE fact.OrderDateKey=data.DateKey AND
      fact.ProductKey=prod.ProductKey AND
      fact.TerritoriesKey=ter.TerritoriesKey AND
      fact.EmployeeKey=emp.EmployeeIdKey AND
      cli.CountryRegionName!='United States'  AND
      prod.Color='Red' AND
     emp.Gender='F' AND
     data.FullDateAlternateKey BETWEEN '2010/06/01' AND '2012/06/01'
GROUP BY ter.Name,prod.ProductCategoryName
ORDER BY ter.name, média
```

Código 3 – Consulta C representada na linguagem SQL

- **Consulta D:**

```
SELECT prod.Name, ter.Name, data.CalendarYear, AVG(emp.SalesQuota) AS
       Quota, SUM(((fact.UnitPricefact-fact.UnitPriceDiscount)*
       fact.OrderQuantity)) AS semDesc, AVG(prod.VendorStandardPrice)
       AS MediaPrecoVendedor
FROM  FactSalesOrders fact, DimProduct prod, DimDate data,
      DimCustomer cli, DimTerritories ter, DimEmployee emp,
      DimShipMethod ship
WHERE fact.OrderDateKey=data.DateKey AND
      fact.ProductKey=prod.ProductKey AND
       fact.TerritoriesKey=ter.TerritoriesKey AND
       fact.EmployeeKey=emp.EmployeeIdKey AND
       fact.ShipMethodKey=ship.ShipMethodKey AND
       ter.CountryRegionName!='Australia'  AND
       ship.Name!='CARGO TRANSPORT 5' AND
       data.FullDateAlternateKey BETWEEN '2005/03/01' AND '2014/03/01'
GROUP BY prod.Name,ter.Name, data.CalendarYear
ORDER BY prod.Name,ter.name
```

Código 4 – Consulta D representada na linguagem *SQL*

4.5 Conclusões

Neste capítulo realizaram-se várias tarefas que servem de suporte aos testes de performance a realizar. A partida para este tema foi dada pela seleção dos sistemas a utilizar, onde a popularidade dos sistemas *MongoDB* e *Cassandra* se contrapõem à experiência do sistema *SQL Server*.

Também foi possível compreender vários fatores inerentes aos processos de desenho de esquemas, em bases de dados *NoSQL*. Tendo sido dado especial relevo às caraterísticas "livres de esquemas" e às implicações que este fator transporta para o desenho de consultas e para os processos de modelação de dados.

Ainda neste capítulo, deu-se início à resposta ao objetivo principal de se estudar a viabilidade da utilização de bases de dados *NoSQL* em armazéns de dados. Essa situação deu-se com a criação do modelo de dados multidimensional e o desenho das consultas na linguagem *SQL*. Estes dois processos irão servir como base de comparação para se conseguir averiguar as diferenças e dificuldades que existem para a realização dos mesmos processos nos dois sistemas *NoSQL* selecionados.

5 Armazéns de Dados em MongoDB

5.1 Introdução

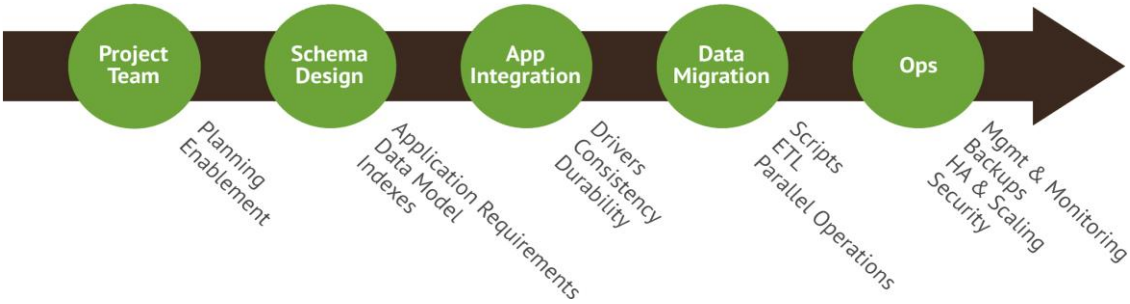

Figura 19 – Mapa para a migração de dados para o sistema *MongoDB*. Retirado de (MongoDB, 2014j)

Este capítulo irá incidir na sua totalidade sobre o sistema *MongoDB*. Nele serão referidos quais os processos e estudos que irão ser realizados, de modo a se conseguir integrar um armazém de dados neste sistema.

A equipa do sistema *MongoDB* definiu uma metodologia que visa migrar com sucesso uma base de dados relacional para uma instância *MongoDB*. Os vários passos que constituem esta metodologia podem ser observados na figura anterior. Este capítulo irá seguir os cinco pontos definidos na mesma. Deste modo, logo após o planeamento e instalação inicial do sistema, irão ser estudados os conceitos inerentes ao desenho de esquemas. Depois de esta tarefa estar concluída será necessário criar uma aplicação capaz de fornecer a ponte entre o armazém de dados *SQL Server* e o novo armazém a ser criado neste sistema. Assim que este processo se encontrar concluído restará apenas proceder à migração dos dados. Nesta dissertação o último

passo do esquema apresentado não será considerado, uma vez que será substituído pelos testes de performance desenhados.

5.1.1 Instalação e configuração

O sistema MongoDB é facilmente instalável numa máquina com ambiente Windows. Basta aceder à secção de transferências do endereço oficial do sistema (MongoDB, 2014g) e transferir a versão mais recente. Como a grande maioria dos sistemas no mundo NoSQL, o MongoDB não possui um interface gráfico, fornecendo apenas um CLI para interação "manual" com o sistema. Por conseguinte, o sistema necessita de ser iniciado via linha de comandos e para iniciar necessita de ser informado da localização onde as instâncias do sistema irão armazenar os dados. Por omissão esta aplicação armazena os dados na localização "\data\db", bastando assim criar a localização (manualmente) se a mesma não existir, também é possível utilizar uma outra qualquer localização para armazenar os dados, bastando para isso definir o atributo "dbpath" aquando da inicialização do sistema.

```
C:\Servidores\mongodb\bin\mongod.exe --dbpath E:\MongoStore\data\db
```

Código 5 – Inicialização do sistema *MongoDB*

Neste tipo de inicialização a linha de comandos onde foi executada a linha anterior tem de permanecer aberta enquanto o sistema *MongoDB* está a ser utilizado. Para contornar este problema e simplificar ligeiramente este processo, deve-se criar um serviço *Window*s que pode ser inicializado automaticamente com o sistema operativo, ou simplesmente parado ou iniciado através da aplicação de gestão de serviços do *Windows.*

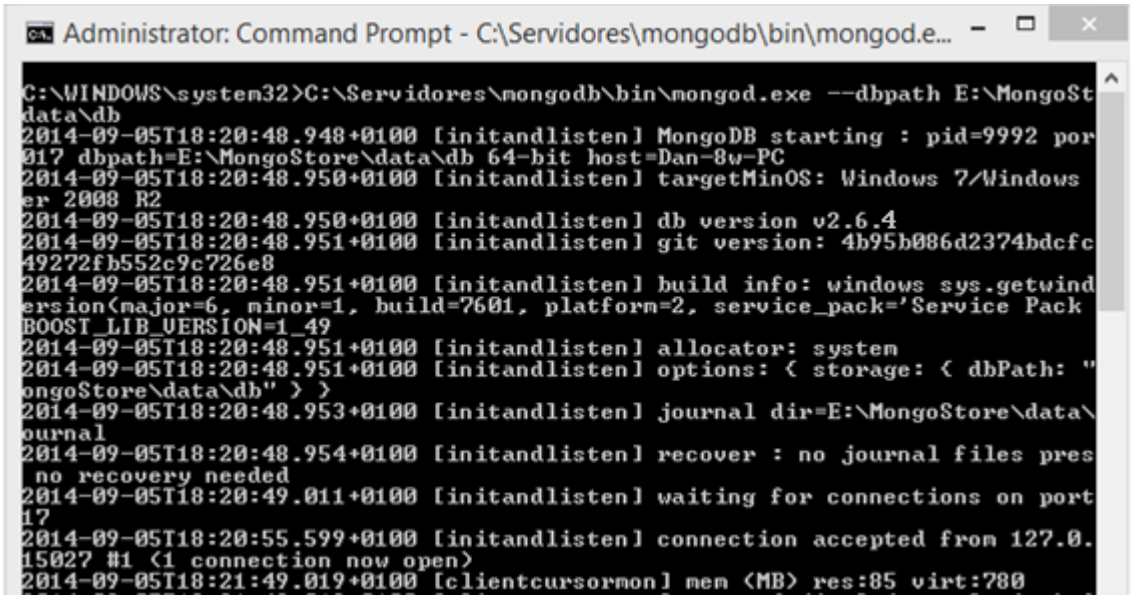

Figura 20 – Inicialização do sistema *MongoDB* e consequentes informações dispostas na consola

5.2 Modelo de dados & Desenho de Esquema

5.2.1 Contextualização

Como já foi referido na secção 4.4 deste documento, existem várias considerações a ter quando se pensa no desenho do modelo de dados de um sistema *NoSQL*. No entanto, neste subponto o foco estará apenas na criação de um esquema capaz de representar um armazém de dados, atendendo às caraterísticas e capacidades do sistema *MongoDB*.

Tabela 4 - Conversão de terminologia relacional para *MongoDB*. Retirado de (MongoDB, 2014j)

Bases de dados Relacionais	***MongoDB***
Base de dados	Base de Dados
Tabela	Coleção
Linha	Documento
Índice	Índice
Junção (*JOIN*)	Embutir documentos ou referenciar

Na Tabela 4 está ilustrada a "conversão" de alguns conceitos utilizados no mundo relacional, para a sua equivalência em *MongoDB*. É importante referir que os conceitos ilustrados não possuem uma equivalência direta, as suas implementações são bastante díspares. O objetivo desta comparação passa por fornecer uma contextualização "hierárquica" das estruturas de armazenamento *MongoDB*. Para se compreender na totalidade a estrutura de armazenamento do MongoDB é necessário ganhar uma maior consciência do que é um "documento" no contexto deste sistema.

5.2.2 JSON, BSON e o Armazenamento Orientado a Documentos

O sistema *MongoDB* é uma base de dados *NoSQL* de armazenamento orientado a documentos. O que são documentos neste contexto? Como já foi ligeiramente abordado na secção 3.5.2, de uma forma pragmática pode-se afirmar que, documentos em *MongoDB* não passam de ficheiros moldados no formato *JSON*, mas armazenados segundo o formato *BSON* (Copeland, 2013).

O formato de troca de dados *JSON* é mais que uma simples e interessante forma de partilhar dados, também é uma excelente forma de os armazenar. Os ficheiros no formato *JSON* são capazes de armazenar dados de uma forma rica e expressiva, como todo o conteúdo de um documento pode ser descrito efetivamente, não há necessidade de especificar qual é a estrutura de um documento antes de este ser criado e submetido para a base de dados (Plugge et al., 2010).

A utilização de documentos baseados em JSON garante ao sistema *MongoDB* o cumprimento da sua filosofia "livre de esquema", visto que um documento pode ter os seus atributos modificados individualmente, sem que isso afete a estrutura de outros documentos na mesma coleção. O armazenamento no formato *BSON* não modifica em nada a maneira como os dados são trabalhados pelo utilizador, mas torna o sistema mais rápido porque a pesquisa e processamento de dados se torna mais fácil (Plugge et al., 2010). Este formato ainda adiciona algumas caraterísticas que não estão disponíveis no formato *JSON*, como o armazenamento de dados binários. Como é baseado no *JSON*, o formato *BSON* também é constituído por um conjunto ordenado de chaves com valores associados (pares chave-valor) (Chodorow, 2013). Um exemplo da estrutura de um documento *BSON* pode ser observado na Figura 21.

Figura 21 – Exemplo de um documento BSON. Retirado de (MongoDB, 2014e)

Todos os documentos armazenados no sistema têm de possuir na sua estrutura uma chave "*_id*". Numa coleção, todos os documentos têm de possuir um valor único para o campo "*_id*", desse modo garantindo que cada documento pode ser inequivocamente identificado dentro de uma coleção (Chodorow, 2013). Para melhor se compreender este elemento pode-se estabelecer um paralelismo com a chave primária das bases de dados relacionais. O valor desta chave pode ser de qualquer tipo de dados, no entanto, por defeito, estará definido como sendo do tipo *ObjectID. O tipo de dados ObjectID* foi desenhado pela equipa do *MongoDB* para ser extremamente leve e garantir que é extremamente fácil gerar novos valores únicos dentro de um ambiente altamente distribuído (Chodorow, 2013). A razão pela qual neste sistema se opta por um novo tipo de dados, suprimindo a escolha pela tradicional chave primária auto incremental, deve-se ao facto de ser bastante difícil sincronizar e garantir que chaves auto incrementadas são únicas quando o armazenamento de uma coleção está distribuído por vários nós. Se, como no caso da figura anterior, não existir um campo "*_id*" no documento a inserir, o mesmo será gerado automaticamente pelo sistema e adicionado ao documento a armazenar (Chodorow, 2013).

5.2.3 Referenciar VS Embutir

Uma caraterística que este sistema partilha com a grande maioria dos sistemas *NoSQL* é o facto de não possuir mecanismos que lhes permitam efetuar a junção (*Joins*) de coleções. Por contraposto ao que acontece nos sistemas relacionais, que vivem num mundo onde impera a lei da normalização, que geralmente resulta num elevado número de tabelas. O estudo deste aspeto da modelação de dados do sistema *MongoDB* é de extrema importância para se conseguir alcançar os objetivos definidos nesta dissertação. Num armazém de dados tradicional,

a tabela de factos possui relações com todas as tabelas dimensionais. Assim, foi necessário aferir-se como devem ser desenhadas ou emuladas relações entre dados num sistema onde não é possível "unir" diferentes tabelas (coleções no contexto MongoDB). O estudo revelou que existem 3 soluções para esta situação: embutir, referenciar ou uma combinação dos dois.

5.2.3.1 Referenciar

Tal como o nome indica, referenciar alude a um processo maioritariamente idêntico ao que acontece nas relações entre tabelas dos sistemas relacionais. No sistema *MongoDB* uma referência para um documento (normalmente) de outra coleção é armazenada como o valor de um atributo num documento. À semelhança do que acontece nos sistemas relacionais, em que se utiliza a chave primária de um registo como referência, no sistema *MongoDB* utiliza-se o elemento "*_id*" como valor a referenciar.

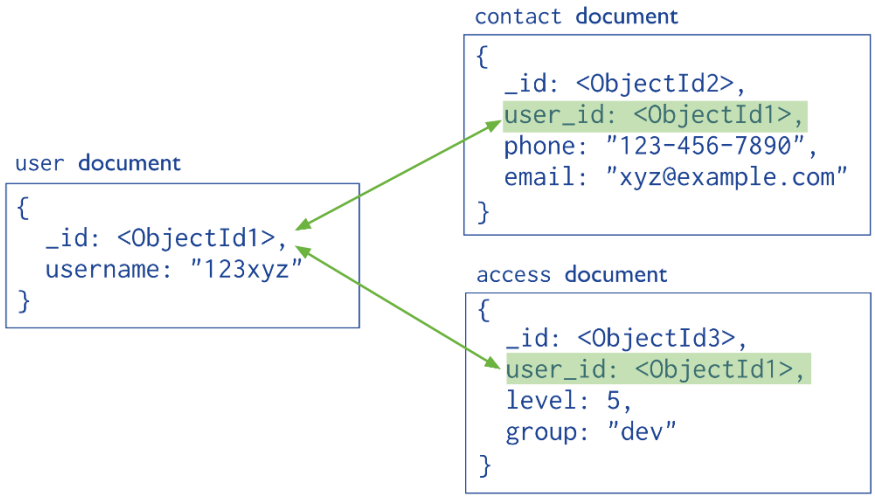

Figura 22- Referências a outros documentos em MongoDB. Retirado de (MongoDB, 2014f)

A utilização deste método de modelação de dados, também conhecido como modelo normalizado (Borland, 2014), possibilita que apenas um documento seja modificado na eventualidade de uma atualização ser levada a cabo, num dos elementos que serve de referência, ao contrário do que acontece no caso de os documentos serem embutidos. Nessa situação todos os documentos que possuíssem o elemento a modificar terão de ser acedidos e modificados (Chodorow, 2013). A grande desvantagem deste modelo é que devido ao facto de o sistema MongoDB não possuir mecanismos que possibilitem a união de coleções, as referências a outros documentos terão de ser resolvidas ao nível da aplicação, desse modo implicando que pelo menos duas consultas terão de ser pedidas ao servidor. Este modelo de dados deve ser essencialmente utilizado (MongoDB, 2014f):

- Quando embutir documentos resultar numa elevada duplicação de dados, sem que isso contribua para um aumento na performance de leitura de dados capaz de suportar as desvantagens induzidas por essa duplicação.
- Para representar relações "muitos-para-muitos" mais complexas.
- Para modelar conjuntos de dados altamente hierárquicos.

5.2.3.2 Embutir

Os esquemas MongoDB que recorrem a documentos embutidos representam relações ao armazenarem todos os dados relacionados no mesmo documento. Como um documento no sistema *MongoDB* é essencialmente um ficheiro *JSON,* que também na sua essência é um conjunto de pares chave-valor, isto possibilita que vetores ou mesmo documentos sejam fornecidos como valor para uma determinada chave. Assim sendo, vários documentos e/ou estruturas podem estar representadas num único documento.

Este modelo de dados MongoDB, também conhecido como modelo desnormalizado (Borland, 2014), implica que todos os documentos de uma coleção necessitem de ser modificados se for necessário alterar alguma informação num dos documentos embutidos, mas por outro lado, será apenas necessário uma *query* para se conseguir alcançar todos os dados relacionados (Chodorow, 2013).

Uma outra vantagem desta abordagem é que mantém os dados relacionados juntos em disco, o que, por sua vez também pode ser um problema. No modelo de documentos embutidos, existe a tendência para os documentos crescerem e ocuparem mais espaço em disco. A consequência imediata é que estes serão mais difíceis de sincronizar por todos os outros nós do sistema distribuído (Chodorow, 2013). O espaço elevado dos documentos também pode perturbar fortemente a performance de escrita do sistema, o que poderá causar problemas de fragmentação (MongoDB, 2014f). Outro aspeto ainda a considerar é que os documentos armazenados no sistema *MongoDB* não podem possuir tamanhos superiores a 16 *megabytes (Borland, 2014)*. É assim bastante importante considerar o tamanho que um documento pode vir a ter quando se considera a utilização de um esquema com documentos embutidos.

Figura 23 – Modelo de dados MongoDB contendo documentos embutidos (MongoDB, 2014f)

De uma forma geral, a modelação recorrendo a documentos embutidos proporciona uma melhor performance para operações de leitura, enquanto a modelação de dados recorrendo a referências proporciona uma melhor performance de escrita (MongoDB, 2014f). É assim bastante importante considerar o ambiente da aplicação que irá interagir com o sistema, antes de se definir o modelo de dados a adotar. Em (Plugge et al., 2010) refere-se que como regra geral deve ser utilizado o modelo de modelação com dados embutidos sempre que possível,

pois este método é bastante mais eficiente e quase sempre bastante viável. Por outro lado, (Chodorow, 2013) propõe um conjunto de fatores que devem pesar na escolha de um destes tipos de modelação. Esses fatores estão ilustrados na próxima tabela.

Tabela 5 – Matriz de casos de utilização dos 2 diferentes modelos de dados em MongoDB

Embutir é melhor para…	Referenciar é melhor para…
Documentos Pequenos	Documentos Grandes
Dados que não mudam regularmente	Dados voláteis
Quando eventual consistência é aceitável	Quando consistência imediata é necessária
Documentos que crescem uma pequena percentagem	Documentos que crescem bastante
Dados que normalmente necessitaram de uma segunda query para serem totalmente alcançados	Dados que serão normalmente excluídos de alguns resultados
Leituras rápidas	Escritas rápidas

5.2.4 A estrutura de um armazém de dados em MongoDB

O estudo levado a cabo nos 3 subpontos anteriores tinha como objetivo garantir que eram adquiridos todos os conhecimentos necessários para se desenhar, corretamente, o esquema de um armazém de dados no sistema MongoDB. Esse estudo contribuiu para a conclusão de que o processo mais simples, para se alcançar esta tarefa, é transformar o esquema em estrela do armazém de dados relacional (Figura 18) num esquema *MongoDB*.

No artigo (Zhao et al., 2013) são propostos alguns métodos para se proceder à modelação de esquemas de dados MongoDB seguindo a filosofia do modelo relacional. Mas infelizmente esse artigo não tem em conta situações como a ausência de mecanismos que possibilitem a "união" de documentos. Para se conseguir alcançar o objetivo (principal) desta dissertação, que visa medir a performance de leitura dos vários sistemas, é necessário que o sistema seja capaz de responder a uma consulta de forma independente. Isto significa que a consulta tem de ser expressa apenas na linguagem de *querying* do sistema *MongoDB*, e que o servidor terá de resolver a mesma sem ajuda adicional. Esta situação implica que não seja possível unir e tratar dados de diferentes coleções/documentos na lógica da aplicação, e por consequência, um modelo de dados *MongoDB* que recorra à utilização de referências está automaticamente descartado. Felizmente existe uma variada documentação apresentada pela equipa do sistema *MongoDB* que visa ajudar novos utilizadores a migrarem para o seu sistema.

Tal como sucede no processo de modelação de armazéns de dados tradicionais, a modelação de relações "muitos-para-muitos" acaba por ser a transformação mais complexa a realizar no desenho de esquemas *MongoDB*, sendo que este tópico é um dos mais dissertados e abordados

na construção de armazéns de dados tradicionais (Song et al., 2001). Apesar de existir documentação e exemplos diversificados de como caraterizar esta situação num esquema *MongoDB*, devido à simplicidade do esquema em estrela do armazém de dados criado apenas serão estudadas conversões de relações "um-para-muitos" e "um-para-um".

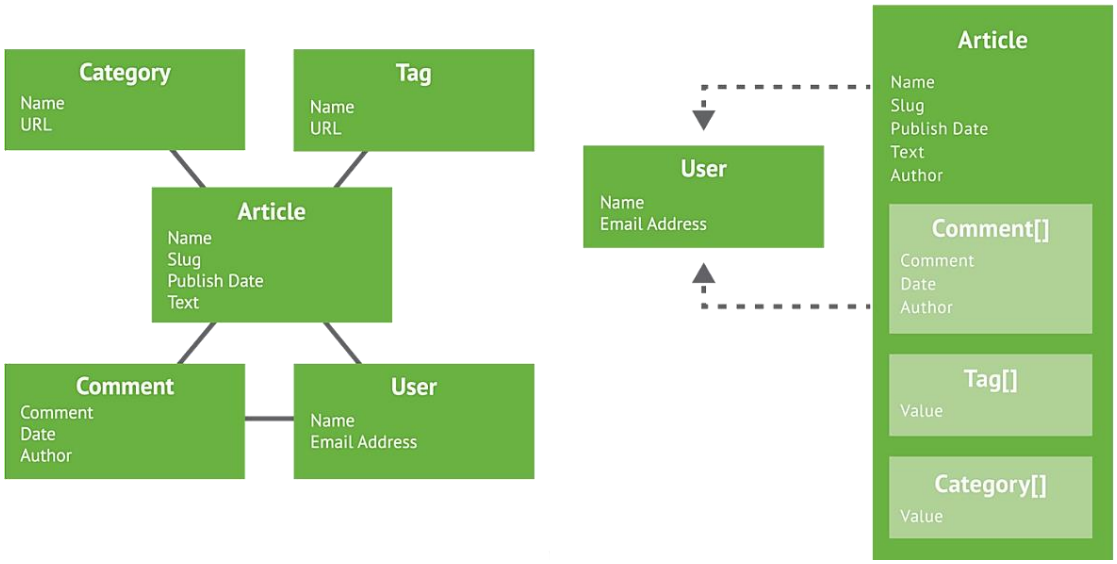

Figura 24 – Modelo de dados relacional, convertido num modelo de dados BSON. (MongoDB, 2014j)

Em (MongoDB, 2014f) é apresentado um padrão a seguir no caso de ser necessário modelar relações "um-para-um" recorrendo a dados embutidos. Segundo a fonte citada, nesta situação deve-se simplesmente embutir o documento secundário na estrutura do documento principal, como no exemplo seguinte:

```
{
 _id: "joe",
 name: "Joe Bookreader"
}

{
 patron_id: "joe",
 street: "123 Fake Street",
 city: "Faketon",
 state: "MA",
 zip: "12345"
}
```

```
{
 _id: "joe",
 name: "Joe Bookreader",
 address:
     {
      street: "123 Fake Street",
      city: "Faketon",
      state: "MA",
      zip: "12345"
     }
}
```

Código 6- Modelação de relações um-para-um no sistema MongoDB

Como é observável o excerto de código anterior está no formato *JSON*, no entanto é bastante fácil extrapolar esta situação para o modelo relacional, porque é muito simples converter (em termos estruturais) uma tabela relacional em documentos. No código apresentado à esquerda é possível visualizar que existem dois documentos com uma relação "um-para-um". O utilizador *Joe* possui uma morada, e essa morada pertence apenas a esse utilizador, desse modo, como pode ser observado no código mais à direita, a morada é embutida como um atributo no documento respeitante a esse utilizador.

No que diz respeito à modelação de relações “um-para-muitos”, é sugerido em (MongoDB, 2014f) que os documentos que se encontrem do lado “muitos” da relação sejam embutidos como um vetor, na estrutura do elemento que se encontra do lado “um” da relação. Esta situação está ilustrada no excerto de Código 7. Nesse excerto, é apresentada à direita a relação do utilizador *Joe* com várias moradas que possui. Esta modelação é levada a cabo, como referido, ao embutir-se as duas moradas no documento relativo a este utilizador, para esse efeito recorrendo a um vetor. Esta situação está apresentada no código apresentado mais à direita.

```
{
 _id: "Daniel",
 name: "Daniel Pereira"
}

{
 patron_id: "Daniel",
 street: "Rua 1º de Dezembro",
 city: "Paredes",
 state: "Porto",
 zip: "4580"
}

{
 patron_id: "Daniel",
 street: "Rua 2º de Dezembro",
 city: "Penafiel",
 state: "Porto",
 zip: "4560"
}
```

```
{
 _id: "Daniel",
 name: "Daniel Pereira"
 addresses:
   [
       {
        street: "Rua 1º de Dezembro",
        city: "Paredes",
        state: "Porto",
        zip: "4580"
       },
       {
        street: "Rua 2º de Dezembro",
        city: "Penafiel",
        state: "Porto",
        zip: "4560"
       }
   ]
}
```

Código 7 – Modelação de relações um-para-muitos em MongoDB. Adaptado de (MongoDB, 2014f)

Com todos os estudos e análises realizados acabou por se tornar claro que, para levar a cabo o processo de transformação do esquema para um esquema MongoDB, seria necessário desnormalizar completamente a estrutura do armazém de dados. Assim e à luz dos exemplos referidos optou-se por se transformar cada linha da tabela de factos num documento. O segundo passo do processo de transformação passou por desnormalizar as relações da tabela de factos com as tabelas dimensionais. Para alcançar esse efeito foi necessário transformar cada linha das tabelas dimensionais num documento. Estes foram então, posteriormente embutidos nos documentos que lhes eram respetivos, resultantes da transformação da tabela de factos. A opção por este tipo de modelação foi validada pelo artigo (Carniel et al., 2012), onde os autores também procederam a uma tarefa semelhante de modelação.

A Figura 55, presente no Anexo D, ilustra a estrutura conceptual do armazém de dados criado no sistema *MongoDB*. Devido ao seu grande tamanho, apenas algumas medidas e atributos dimensionais estão ilustrados na figura. A estrutura completa de um documento *BSON*, pertencente ao armazém de dados criado, pode ser consultada no Anexo E deste documento.

5.3 Programação e Migração de dados

Durante o estudo levado a cabo não foram encontradas ferramentas capazes de migrar os dados prwsentes no armazém de dados *SQL Server* para o sistema *MongoDB*, respeitando o esquema *BSON* criado. Assim, foi necessário recorrer a uma linguagem de programação para se conseguir realizar este processo. Como referido, a linguagem de programação utilizada foi a linguagem *C#.* A equipa do sistema *MongoDB* disponibiliza um driver e uma API *C#* bastante bem documentada, o que torna muito simples programar com foco neste sistema.

Para se poder interagir com o sistema *MongoDB* através da linguagem *C#* e o consequente *IDE Visual Studio* é necessário realizar a transferência do driver *C#* através do gestor de pacotes *Nuget*[12]. É possível proceder à instalação do pacote percorrendo manualmente os repositórios ou simplesmente executando a seguinte linha na consola *Nuget*: `Install-Package mongocsharpdriver.`

Um dos pontos mais importantes a considerar quando se planeia a migração de dados para um sistema diferente é o tipo de dados suportado pelo sistema de destino. Na maioria dos casos é necessário efetuar algum tipo de conversão/mapeamento de dados entre os dois sistemas. O driver *C#* do *MongoDB* facilita muito esta tarefa.

Tabela 6 – Tipos de dados suportados em documentos BSON. Adaptado de (MongoDB, 2014a)

Tipos de dados suportados em documentos *BSON* do sistema *MongoDB*		
Double	*String*	Objeto
Array	Dados Binários	Indefinidos
ObjectID	Booleanos	Data
Null	Expressões regulares	*JavaScript*
Símbolo	JavaScript com âmbito (scope)	Inteiro de 32-*bits*
TimeStamp	Inteiro de 64-*bits*	Chave Mínima e Chave Máxima

Como Já foi amplamente referido, os documentos armazenados em MongoDB têm ser serializados no formato *BSON*. Na Tabela 6 é possível visualizar os tipos de dados suportados em documentos deste formato. O que é observável desta tabela é o facto de apenas existir um tipo de dados para representar valores numéricos não inteiros, o que contrasta com os 3 tipos de dados (*real*, *decimal*, *money*) utilizados pelo esquema do armazém de dados *SQL Server*. Este problema é bastante bem ultrapassado pelo driver utilizado, que consegue identificar e converter automaticamente os dados do tipo "*real*" e "*Money*" para o tipo *double* do formato *BSON*. Infelizmente esta situação não se repete no que diz respeito aos dados do tipo *decimal.* Sendo que uma pré-conversão foi necessária para que o driver os interpretasse. Um outro

[12] É um gestor de pacotes livre e de fonte aberta desenhado para operar com as linguagens da plataforma *.NET*. É distribuído como uma extensão do *visual Studio*. Desde 2010 evoluiu para um grande ecossistema de ferramentas e serviços.

problema foi encontrado ao verificar que os atributos do tipo data no sistema *MongoDB*, estavam todos algumas horas atrasados em relação ao mesmo atributo na base de dados *SQL Server*. Um estudo consequente levou à informação de que os dados do tipo *DateTime* armazenados no *MongoDB*, são guardados no fuso horário *UTC*, por consequência é necessário indicar ao compilador que aquele valor já se encontra nesse fuso horário, o contrário resultará numa conversão errónea e numa data incorreta adicionada à base de dados.

Um terceiro e último problema surgiu neste processo, o tratamento de valores *null*. O conceito de *null* em *MongoDB* é bastante diferente do que acontece nos sistemas relacionais. Devido à sua flexibilidade de esquema para considerar um elemento *null* basta omiti-lo do código do documento a inserir. Documentos da mesma coleção podem possuir estruturas diferentes e documentos podem ser posteriormente atualizados de modo a serem adicionados ou removidos atributos da sua estrutura. No entanto, existem situações específicas em que pode ser necessário inserir um valor *null* num atributo. No caso deste projeto, por exemplo, são serializados cerca de 5 milhões de linhas. Nos dados contidos nessas linhas estão presentes alguns valores *null* (propositadamente), sendo bastante difícil criar programaticamente um ciclo iterativo que contemple a possível existência de um valor *null* em cada uma das 180 colunas do armazém de dados. Situações como esta são contempladas pelo sistema MongoDB, através to tipo de dados *BSONull*. Assim também foi necessário converter atributos *DBnull* para *BSONnull*.

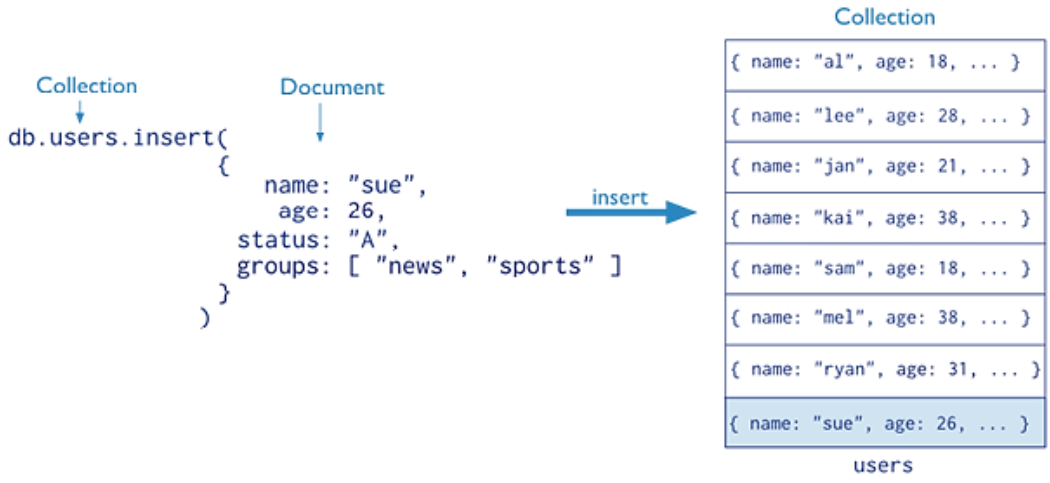

Figura 25 –O processo de inserção de um documento numa coleção *MongoDB*. (MongoDB, 2014e)

Antes de se proceder a uma pequena exemplificação com código, é necessário referenciar os tutoriais *C#* disponíveis pela equipa do *MongoDB* (MongoDB, 2014b). Para aprender a utilizar este sistema basta seguir a excelente documentação fornecida. Devido ao grande número de atributos, não é possível apresentar aqui o código completo do processo de inserção de um documento no armazém de dados MongoDB. Deste modo, serão aqui fornecidos exemplos de código que recorrem a apenas alguns atributos presentes no armazém de dados. A linguagem *C#* segue uma filosofia de programação orientada a objetos, por consequência existe uma hierarquia de classes e métodos a respeitar. Por questão de simplicidade, e para não tornar a leitura do código apresentado em algo complexo e fastidioso, apenas serão apresentadas as

linhas de código mais relevantes, partindo do princípio que o leitor conseguirá enquadra-las sem problema na estrutura, corretamente definida, de uma aplicação *C#.*

```
using MongoDB.Bson;
using MongoDB.Driver;
using MongoDB.Driver.Builders;

//Estabelecer uma ligação ao servidor MongoDB local
private static string connectionString = "mongodb://localhost";
private static MongoClient client = new MongoClient(connectionString);
private static MongoServer server = client.GetServer();

//Definir qual é a Base de dados a utilizar no sistema MongoDB
private static MongoDatabase database =
server.GetDatabase("DissertacaoDimsAW");
//Definir qual é a Coleção a utilizar no sistema MongoDB
private static MongoCollection collection =
database.GetCollection("DimsAndFacts");
```

Código 8 – Excerto C# para efetuar a ligação a um servidor MongoDB local

As primeiras linhas do código anterior servem para referenciar os *DLLs* contidos nos pacotes instalados. O segundo conjunto de linhas representa o código necessário para estabelecer uma ligação cliente-servidor, da aplicação para o sistema *MongoDB*. O terceiro bloco de código limita-se a definir qual será a base de dados e coleção a utilizar no processo de inserção. O excerto de código anterior representa o mínimo de código *C#* necessário para realizar qualquer tipo de interação com um servidor *MongoDB*. Na programação em *C#* orientada para o sistema *MongoDB* não é necessário abrir ou encerrar ligações ao servidor, estando o *driver* inteiramente encarregue de controlar essas tarefas através do seu conjunto de ligações (*connection pool*). O programador pode fechar uma ligação manualmente, mas o mesmo não é recomendado, visto que todas as ligações presentes no *connection pool* serão perdidas (MongoDB, 2014b).

```
collection.Insert(new BsonDocument {
    {"OrderQuantity",BsonValue.Create(15)},
    {"TotalPrice",BsonValue.Create(Convert.ToDouble(124.32m))},
    {"DimCustomer",new BsonDocument {
                {"CustomerTitle",BsonValue.Create(BsonNull.Value)},
                {"CustomerFirstName",BsonValue.Create("Daniel")}
    }},
    {"DimDate",new BsonDocument {
                {"FullDateAlternateKey", BsonValue.Create(DateTime.
                            SpecifyKind( DateTime.Now, DateTimeKind.Utc))}
    }}
});
```

Código 9 – Excerto de código *C#* para inserir um documento no sistema *MongoDB*

O exemplo apresentado acima destina-se a emular a inserção de um documento que possui uma estrutura (parcialmente) parecida com aquela do esquema desenhado. Como é observável, são criados dois documentos (*DimCustomer* e *DimDate*) que são desde logo embutidos no documento principal, este contendo duas medidas. Com este exemplo tentou-se replicar os 3 problemas encontrados no processo de inserção, descrito anteriormente. Como referido, não

é necessário criar previamente estruturas e tipos de dados para documentos, basta passar os valores ao driver *C#,* que este encarrega-se de mapear um tipo de dados para cada elemento. Outro facto a anuir da figura é que não está presente o campo "*_id*", assim atribui-se ao sistema a responsabilidade de criar um identificador único universal para cada registo. O documento inserido, resultante da execução do código anterior, pode ser visualizada na figura seguinte.

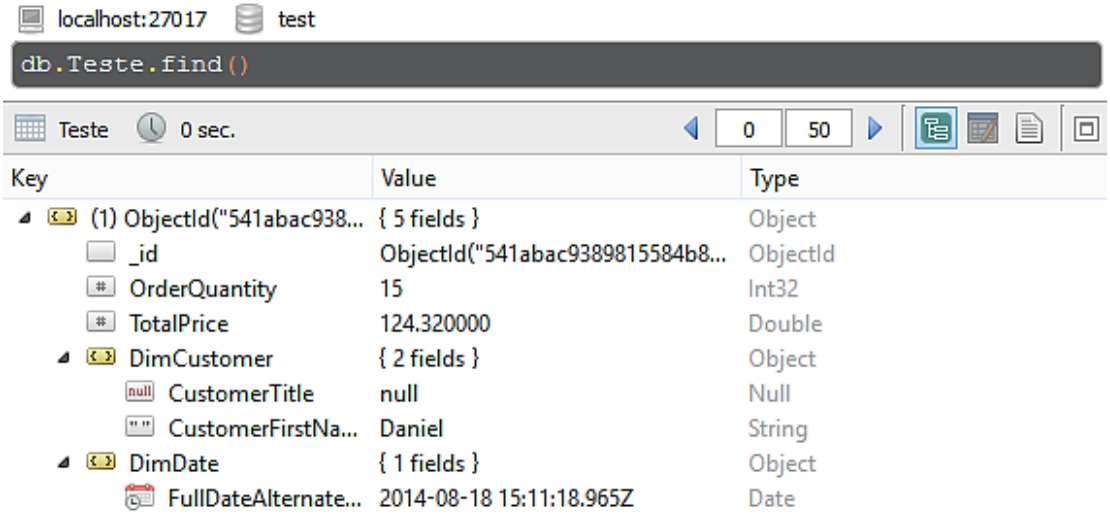

Figura 26 – Representação em formato árvore do documento BSON inserido com o código anterior

É bastante importante salientar que este código é apenas um exemplo. O programa criado para realizar a migração de dados necessitava também de tratar os 5 milhões de linhas resultantes da consulta ao servidor *SQL Server.* Este processo realizado numa máquina com recursos de memória, disco e processador insuficientes tornou-se num desafio bastante interessante. Conceitos relativos a *threading* e outras pequenas modificações tiveram de ser introduzidos, de modo a conseguir tratar-se o elevado volume de dados.

É possível melhorar a performance de escrita de dados no sistema *MongoDB* quando se recorre ao método de inserção em lotes (*batch insert*). Como pode ser observado no excerto de código 10, são criados 2 documentos que são adicionados através do método referido. Este foi o método adotado para inserir novos documentos no armazém de dados *MongoDB*. Existe ainda um outro método de inserção que pode ser utilizado em conjunto com o *driver C#:* a serialização de classes. Apesar de este ser um método bastante interessante para efetuar operações sobre dados em *C#,* acabou por ser preterido, devido à complexidade que adiciona à escrita de código quando conjugado com a conversão de 180 colunas.

```
var BatchDeDocumentos = new List<BsonDocument>();

BatchDeDocumentos.Add(new BsonDocument {
    {"OrderQuantity",BsonValue.Create(15)},
    {"TotalPrice",BsonValue.Create(Convert.ToDouble(124.32m))},
    {"DimCustomer",new BsonDocument {
                {"CustomerTitle",BsonValue.Create(BsonNull.Value)},
                {"CustomerFirstName",BsonValue.Create("Daniel")}
    }},
});

BatchDeDocumentos.Add(new BsonDocument {
    {"OrderQuantity",BsonValue.Create(225)},
    {"TotalPrice",BsonValue.Create(Convert.ToDouble(678.56m))},
    {"DimCustomer",new BsonDocument {
              {"CustomerTitle",BsonValue.Create("Sr.")},
              {"CustomerFirstName",BsonValue.Create("dAnIeL")}
    }},
});

collection.InsertBatch(BatchDeDocumentos);
```

Código 10 – Excerto de código *C#* representativo de uma inserção em lote no sistema *MongoDB*

5.4 Consultas em MongoDB

A linguagem para realização de consultas do sistema *MongoDB* segue, como todas os comandos deste sistema, uma estrutura *JavaScript*. Consultas simples são facilmente mapeadas para a linguagem *SQL*, como pode ser observado na Tabela 7. Por outro lado, consultas mais complexas, com recurso a funções de agregação, tomam um percurso não trivial para um especialista na linguagem *SQL*. O facto de não existir um sistema visual, capaz de ajudar ao processo de construção de consultas, também dificulta bastante a situação. Os comandos a executar têm de ser escritos praticamente caráter a caráter, com umas pequenas ajudas prestadas pelos interfaces gráficos, como o *RoboMongo*. Estas aplicações são bastante úteis para validar as consultas criadas, mas no caso de consultas complexas, não são capazes de melhorar em nada o processo.

Tabela 7 – Equivalência entre consultas *SQL* e *MongoDB*. Adaptado de (MongoDB, 2014c)

SQL	MongoDB
SELECT * FROM users	db.users.find()
SELECT name, age FROM users WHERE age =33	db.users.find({age: 33}, {name: 1, age: 1, _id:0})
SELECT * FROM users WHERE age > 33 AND age < 40	db.users.find({age: {$gt: 33, $lt: 40}})
SELECT * FROM users WHERE age = 33 OR name = 'Bob'	db.users.find({$or:[{age:33}, {name: "Bob"}]})
SELECT DISTINCT name FROM users	db.users.distinct("name")
SELECT COUNT(*) FROM users	db.users.count()
SELECT COUNT(*) FROM users WHERE AGE > 30	db.users.find({age: {$gt: 30}}).count()

A construção das consultas a realizar no sistema *MongoDB*, revelou-se um desafio interessante. Apesar de não ser extremamente complexo, o processo levou a várias leituras e releituras das páginas disponibilizadas pela equipa do sistema *MongoDB* (MongoDB, 2014d).

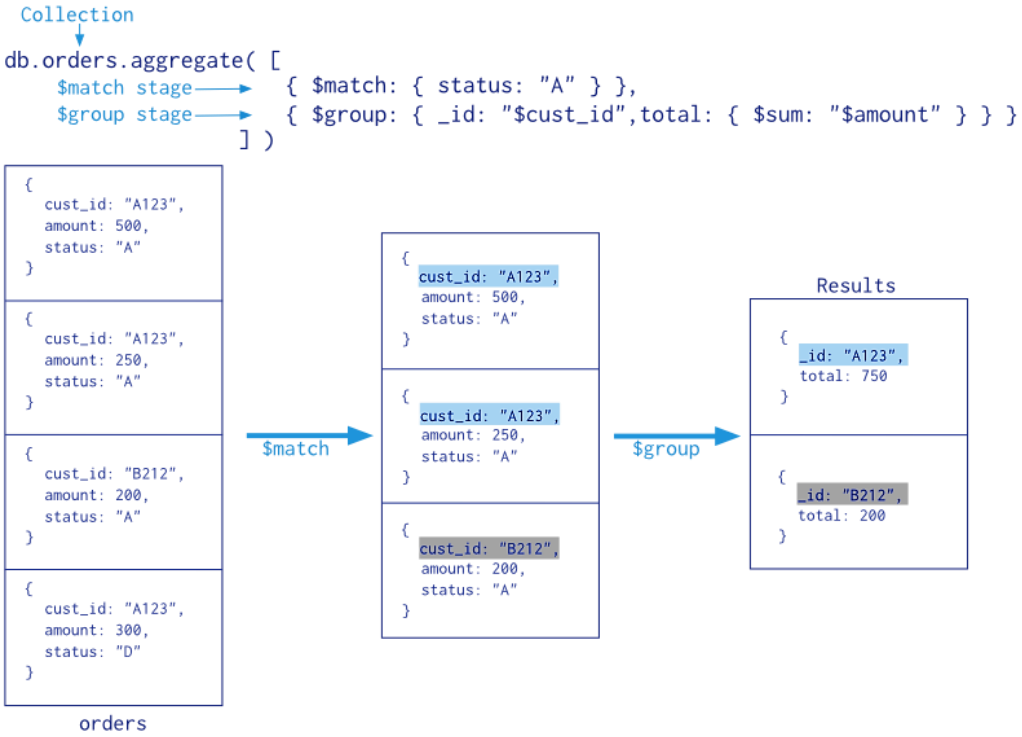

Figura 27 – Os estágios de execução de uma consulta de agregação MongoDB. (MongoDB, 2014h)

Uma surpresa encontrada no estudo levado a cabo é o facto de que, tal como na linguagem *SQL*, é possível realizar a mesma consulta de formas diferentes. Esta descoberta levou a alguma confusão, os exemplos aparecem em locais e situações diferentes e a documentação não elucida qual dos métodos deve ser utilizado. Um exemplo para esta situação vai ser fornecido na consulta A. Nas outras consultas assistiu-se ao mesmo problema, sendo que a consulta com o aparente tempo de execução mais rápido foi a selecionada.

5.4.1 Consulta A

5.4.1.1 Versão 1

```
db.DimsAndFacts.aggregate([
                    { $match : { 'DimDate.CalendarYear':2008,
                                 OrderQuantity:{ $gte:2, $lte:10 } } },
                    { $group : { _id:{}, total:{ $sum :{ $multiply:
                                 ["$UnitPriceDiscount","$OrderQuantity"]}}}}
                          ])
```

Código 11 – Estrutura da consulta A, a ser executada no sistema *MongoDB*

5.4.1.2 Versão 2

```
db.runCommand({
                group: {
                        ns: 'DimsAndFacts',
                        key: {},
                        cond: { 'DimDate.CalendarYear':2008,
                                OrderQuantity:{ $gte:2, $lte:10 } },
                        $reduce: function ( atual, resultado ) {
                                resultado.total +=(atual.UnitPriceDiscount*
                                                   atual.OrderQuantity);
                                },
                        initial: { total : 0 }
                       }
            })
```

Código 12 – Uma outra versão da consulta A. Este excerto foi preterido em relação ao código anterior

5.4.2 Consulta B

```
db.DimsAndFacts.aggregate([
                     { $match :
                     { 'DimProduct.ProductCategoryName':"Accessories",
                      'DimShipMethod.ShipMethodName':"CARGO TRANSPORT 5"}},
                     { $group : { _id:{ano:'$DimDate.CalendarYear',
                                  nome:'$DimProduct.ProductName'},
                                  total:{ $sum : "$Freight" }}},
                     { $sort : { _id: 1 }   ])
```

Código 13 – Estrutura da consulta B, a ser executada no sistema MongoDB

5.4.3 Consulta C

```
db.DimsAndFacts.aggregate([
                  { $match :
                      { 'DimCustomer.CustomerCountryRegionName':{$ne:'United
                         States'}, 'DimProduct.ProductColor':"Red",
                         'DimEmployee.EmployeeGender':"F",
                        'DimDate.FullDateAlternateKey':{$gte: ISODate("
                         2010-06-01T00:00:00.000Z"), $lte: ISODate("2012-
                         06-01T00:00:00.000Z")}}},
                  { $group :
                      { _id:{territorio:'$DimTerritories.TerritoryName',
                            categoria:'$DimProduct.ProductCategoryName'
                          }, média:{ $avg : "$TotalPrice" }}},
                  { $sort : { _id: 1 } } ])
```

Código 14 – Estrutura da consulta C, a ser executada no sistema *MongoDB*

5.4.4 Consulta D

```
db.DimsAndFacts.aggregate([
            { $match :
               { 'DimTerritories.TerritoryCountryRegionName':
                 {$ne:'Australia'},DimShipMethod.ShipMethodName
                 ':{$ne:"CARGO TRANSPORT 5"},
                  'DimDate.FullDateAlternateKey':{$gte: ISODate("2005-
                   03-01T00:00:00.000Z"), $lte: ISODate("2014-03-
                   01T00:00:00.000Z")}}},
            { $group :
                { _id:{ produto:'$DimProduct.ProductName',
                        territorio:'$DimTerritories.TerritoryName',
                       Ano:'$DimDate.CalendarYear'},
                  Quota:{ $avg : '$DimEmployee.EmployeeSalesQuota' },
                  semDesc:{ $sum :{ $multiply:[ { $subtract:
                           ["$UnitPrice","$UnitPriceDiscount"]},
                            "$OrderQuantity"]}},
                  MediaPrecoVendedor:{ $avg :
                          '$DimProduct.ProductVendorStandardPrice' } }},
            { $sort : { _id: 1 } }
        ])
```

Código 15 – Estrutura da consulta D, a ser executada no sistema *MongoDB*

5.5 Conclusões

Neste capítulo realiza-se uma investigação que abrange apenas o sistema *MongoDB*. É bastante simples entrar e testar o mundo deste sistema. Quando se pensa no desenho de esquemas, a situação torna-se mais complicada. Não é exasperantes desenhar um esquema de base de dados para o sistema *MongoDB*, mas causa alguns desafios. Por outro lado, o desenho de consultas, é um processo bastante complexo. A linguagem utilizada, afasta-se bastante da sintaxe SQL. Implica repensar a metodologia utilizada no desenho de análises. Apesar de não ser extremamente difícil, este foi o processo que mais complexidade reteve. Em comparação com o processo de desenho e integração de um armazém de dados num sistema relacional, apenas o processo de realização de consultas trará dificuldades a novos utilizadores deste sistema. Pensa-se que qualquer novo utilizador não terá problemas em realizar qualquer dos outros processos, depois de algum estudo.

6 Armazéns de Dados em *Cassandra*

6.1 Introdução

Este capítulo irá incidir na sua totalidade sobre o sistema *Cassandra*. Nele serão referidos quais os processos e estudos que irão ser realizados, de modo a conseguir implementar-se um armazém de dados neste sistema. Neste capítulo serão seguidos os mesmos pontos apontados pela equipa do sistema *MongoDB* no início do capítulo anterior.

Ao contrário do que sucedeu no esquema *MongoDB,* vários problemas surgiram com o sistema Cassandra. Os estudos elaborados e os passos adotados para ultrapassar esses obstáculos serão documentados ao longo deste capítulo.

6.1.1 Instalação e configuração

O processo de instalação do sistema *Cassandra* num ambiente *Windows* não é tão simples como sucede no caso do sistema *MongoDB*. O primeiro ponto a dificultar este processo é o facto de não existir nenhuma distribuição *Cassandra* desenvolvida para este sistema operativo. Existem no entanto duas soluções para esta situação. A primeira solução passa por efetuar a transferência da distribuição *Cassandra* destinada aos sistemas *Unix*, modificar manualmente os ficheiros de configuração do sistema e ainda proceder manualmente à instalação de um interpretador *Python*. A segunda opção implica apenas a instalação do sistema *DataStax Community* (DataStax, 2014e). A *Datastax* é uma organização que se dedica à disponibilização de soluções informáticas, que têm por objetivo ajudar as organizações a conseguirem tirar benefícios de dados *Big Data* (DataStax, 2014a). A solução mais importante comercializada por esta empresa, o sistema *DataStax Enterprise*, possui uma base de dados *NoSQL*, construída em cima do sistema *Cassandra*, e ainda várias aplicações que visam gerir, supervisionar e tirar

conhecimento dos dados armazenados. O *DataStax Community* é uma versão simplificada dessa solução e com uma utilização livre de custos para fins não comerciais.

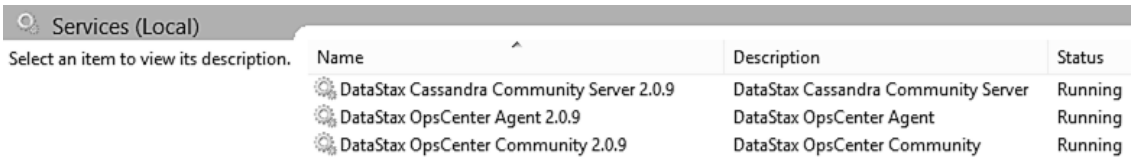

Figura 28 - Serviços Windows instalados com o sistema DataStax Community

Devido à simplicidade e aos benefícios a adquirir, seguiu-se a segunda opção referida no parágrafo anterior. É importante salientar que o *DataStax Community* possui uma versão completamente integral e não modificada do sistema Cassandra que simplesmente facilita a vida do utilizador ao envolver tudo num "pacote" de fácil instalação. Um outro benefício obtido com esta opção é a inclusão na instalação de uma ferramenta de monotorização do sistema e ainda a configuração automática do sistema Cassandra como um serviço *Windows*, ao contrário do que sucede no sistema *MongoDB*. Assim, o sistema pode ser iniciado ou parado sem ser preciso recorrer à execução de comandos numa consola. No entanto, tal como acontece no sistema *MongoDB* e na maioria dos outros sistemas *NoSQL*, este e todos os outros processos de interação com o sistema podem ser sempre realizados via linha de comandos.

6.2 Modelo de dados & Desenho de Esquema

6.2.1 Contextualização

Todas as preocupações e processos levados a cabo no processo de desenho do esquema do sistema MongoDB são aplicáveis ao sistema *Cassandra*. No entanto, vários outros problemas e dúvidas surgiram neste ponto da investigação. Esta situação deve-se principalmente ao facto de o sistema *Cassandra* ter procedido a uma importante modificação no método principal de interação com o sistema. O sistema Cassandra era usualmente acedido através de um interface *thrift*[13], até que no último trimestre de 2012 a equipa de desenvolvimento do sistema, publicou a terceira versão do *CQL (Cassandra Query Language*) (Apache, 2014c). Esta terceira versão tornou-se então no método standard para se proceder a interações com o sistema. Esta mudança trouxe para a cena do sistema Cassandra uma simplicidade muito maior na interação com o sistema. Por outro lado, esta mudança implicou uma nova filosofia de modelação de dados. O *CQL* 3 aproxima-se muito da linguagem *SQL* que, por sua vez, torna mais invisível para o utilizador a forma como os dados são fisicamente armazenados, o que não acontecia no antigo interface *thrift*. Esta situação implicou uma mudança ligeiramente abrupta pelo que

[13] O *Apache Thrift* é uma linguagem de definição de interfaces e um protocolo binário inicialmente desenvolvida pelo *Facebook*.É utilizada para definir e criar serviços para várias linguagens.

mesmo os mestres do sistema *Cassandra* necessitaram de voltar a estudar e a repensar as suas interações com o sistema.

Do ponto de vista de quem investiga este sistema pela primeira vez em 2014, não existem os referidos problemas de transição, pois o *CQL3* é o método de acesso estabelecido. Sendo o interface *thrift* já considerado obsoleto pela maioria das equipas de desenvolvimento (Lebresne, 2012). Um problema que existe, no entanto, para quem investiga pela primeira vez atualmente é a desinformação. Existem vários artigos, livros e páginas web a fornecerem informação de como se deve modelar esquemas de dados para o sistema Cassandra, segundo o antigo interface *thrift*. Estes artigos misturam-se com aqueles correspondentes à linguagem *CQL3* e alguma confusão foi adicionada a uma modelação de dados já "estranha" por natureza. Nesta investigação não foram consideradas alternativas a problemas causados pela utilização da linguagem *CQL3*, aquando do desenho do esquema de dados. O interface *Thrift* não será referenciado ou estudado nesta investigação. Tal como sucedeu com o sistema MongoDB, a melhor forma de iniciar o estudo do sistema *Cassandra* é comparando os seus conceitos com os das bases de dados relacionais.

Tabela 8 - Conversão de terminologia relacional para *Cassandra*. Retirado de (Korla, 2013)

Bases de dados Relacionais	Cassandra
Base de dados	*KeySpace*
Tabela	Família de Colunas
Chave primária	Chave de linha
Linha	Linha

6.2.2 CQL3 e o drama da utilização de famílias de colunas

Exatamente como no caso do sistema *MongoDB*, neste sistema não é possível realizar a união de dados (*JOINs*) de diferentes famílias de colunas. Pelos mesmos motivos salientados no caso do *MongoDB* é imperativo para os objetivos a alcançar com esta investigação que o sistema seja capaz de responder a consultas complexas com uma só *query*, sem que qualquer lógica da consulta seja levada a cabo por parte do código da aplicação. O estudo levado a cabo, revelava mais uma vez que a solução para esta situação será desnormalizar o armazém de dados relacional de modo a ser possível inserir os dados num esquema *Cassandra* (Parthasarathy, 2013). No estudo elaborado sobre as bases de dados *NoSQL* do tipo de armazenamento famílias de colunas é observável a possibilidade de inserir colunas dentro de outras colunas, criando super colunas, como exemplificado na Figura 29. À semelhança do que acontece no sistema *MongoDB*, com a utilização de super colunas seria possível fornecer uma estrutura hierárquica ao esquema, apesar da desnormalização total dos dados.

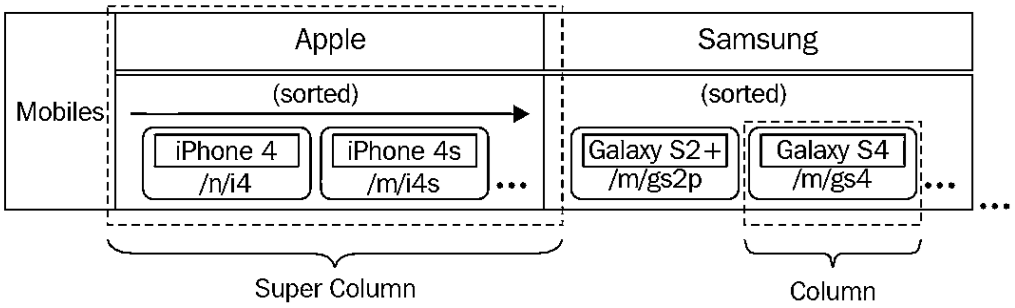

Figura 29 – Duas super colunas inseridas numa família de colunas. Retirado de (Neeraj, 2013)

Após um estudo aprofundado, foi possível concluir que já não é possível desenhar um esquema como aquele presente na figura anterior. Como foi brevemente mencionado, a mudança efetiva e total de um interface *Thrift* para a linguagem *CQL3* alterou várias fatores na interação com o sistema *Cassandra*, incluindo a maneira como é modelado o esquema. Na documentação oficial do sistema Cassandra versão 1.1 (a versão atual é a versão 2.1) é referido o seguinte sobre a utilização de super colunas: "Não utilize super colunas" (DataStax, 2012). Segundo esta fonte, este método representa um processo de modelação de dados obsoleto, que foi estruturado para satisfazer as necessidades de um caso específico e que não se aplica aos casos de utilização da maioria dos utilizadores. Para além de poder conduzir a graves problemas de performance. Na fonte citada ainda se refere um outro ponto bastante importante: "Adicionalmente, as super colunas não são suportadas em *CQL3*".

O primeiro plano para a criação de um esquema capaz de suportar um armazém de dados no sistema Cassandra foi embaraçosamente derrotado pelas imposições da linguagem *CQL3*. Um estudo mais profundo teve de ser realizado.

6.2.3 Colunas e Famílias de colunas em CQL3

Como a utilização de super colunas já não é suportada, a equipa responsável pelo sistema Cassandra equipou a linguagem *CQL3* com um novo componente de modelação de dados, as colunas compostas. Esta nova estrutura é capaz de reproduzir algumas das funcionalidades anteriormente disponibilizadas pelas super colunas. Antes de se investigar este novo componente é necessário compreender também todos os outros componentes de modelação do sistema *Cassandra*. Em Cassandra as famílias de colunas podem ser de dois tipos (Parthasarathy, 2013):

6.2.3.1 Tipos de famílias de colunas

Fixas (estáticas): Bastante parecido com as colunas presentes numa tabela de uma base de dados relacional. As colunas são definidas, validadas e inseridas aquando do processo de criação de uma família de colunas. Índices secundários podem ser construídos sobre estas colunas. Esta situação implica que novas linhas inseridas na base de dados possuam apenas colunas definidas no esquema criado. A utilização deste tipo de colunas traz especiais vantagens

nos casos em que os dados são partilhados por várias aplicações e os mesmos necessitam de ser validados antes de serem inseridos.

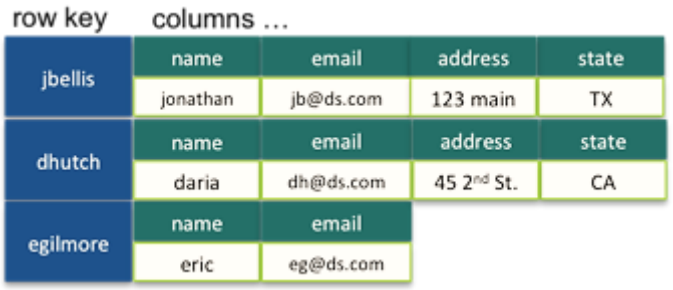

Figura 30 – Exemplo de uma família de colunas estática. Retirado de (DataStax, 2012)

Dinâmicas: Seguindo uma filosofia livre de esquema, o sistema cassandra contempla um cenário em que colunas podem ser criadas, sem que se conheça o nome que irão tomar. Para os utilizadores vindos do mundo relacional, esta situação parece bastante estranha. Contudo, pode ser bastante vantajosa. Por exemplo, numa situação em que se pretende armazenar quais os *blogs* seguidos por um utilizador como o nome de cada coluna, para se armazenar como valor de cada coluna, o número de vezes que o utilizador visitou esse *blog*. Todos os utilizadores vão possuir diferentes *blogs* seguidos, logo é impossível definir os nomes das colunas aquando da criação do esquema. Uma ilustração deste exemplo pode ser visualizada na figura seguinte.

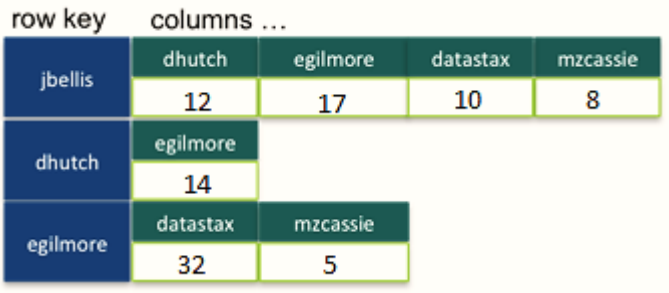

Figura 31 – Exemplo de uma família de colunas dinâmica. Adaptado de (DataStax, 2012)

É preciso considerar que todo este dinamismo é restringido pela linguagem *CQL3*. Não será exemplificado algo mais sobre este tipo de família de coluna, pois este está claramente excluído de um tipo de estrutura a utilizar no armazém de dados *Cassandra*. O armazém de dados possui sempre os mesmos campos em todas as linhas, o dinamismo e a complexidade que causaria não se enquadram ao caso de estudo abordado.

6.2.3.2 Tipos de colunas

As famílias de colunas (como o nome indica) são colunas compostas por outras colunas. Existem 4 tipos de colunas que podem ser utilizados para albergar dados num dos dois tipos de famílias de colunas (Parthasarathy, 2013).

Coluna padrão: Uma coluna padrão não passa de um mero tuplo de dados e representa a estrutura mais simples que pode ser armazenada no sistema Cassandra. O tuplo é constituído

pelo nome da coluna, que pode ser definido *a priori* ou posteriormente de uma forma dinâmica, pelo valor da coluna e por um *timestamp* de quando o valor foi inserido. O valor do timestamp é utilizado pelo sistema *Cassandra* para efeitos de resolução de conflitos. Dos 3 campos referidos, apenas o nome da coluna e o timestamp são obrigatórios.

Colunas Compostas: Este é o método proposto pela equipa do sistema *Cassandra* para substituir a utilização de super colunas, visto que não é suportada pela linguagem *CQL3*. O motor de armazenamento do sistema *Cassandra* utiliza colunas compostas, para armazenar linhas agrupadas. Isto significa que linhas que possuam a mesma chave de particionamento são agrupadas, a um nível físico, como sendo uma única linha. A utilização deste tipo de colunas acaba por constituir um método bastante interessante, porque a diferença que causa no esquema de dados é virtualmente invisível para o utilizador, esta ocorre apenas ao nível do armazenamento físico dos dados. Isto não significa que a sua utilização não seja útil, existem *queries* em Cassandra cuja performance melhora exponencialmente e outras *queries* que só conseguem ser respondidas corretamente se forem utilizadas colunas compostas.

Partition key | Remaining key

user_id	tweet_id	author	body
gmason	1765	phenry	Give me liberty or give me death
gmason	1742	gwashington	I chopped down the cherry tree
ahamilton	1797	jadams	A government of laws, not men
ahamilton	1742	gwashington	I chopped down the cherry tree

Figura 32 – Exemplo de colunas compostas (Disposição visual). Retirado de (DataStax, 2012)

Um exemplo desta situação pode ser apresentado com a observação da figura 32 e da figura 33. Na figura anterior está ilustrada uma família de colunas que possui colunas compostas. Como referido, a um nível visual não é possível perceber qualquer diferença entre esta família de colunas e uma que não possua colunas compostas. As colunas compostas são formadas segundo uma chave de partição fornecido aquando da criação do esquema. Juntamente com a chave de partição podem ser fornecidas outras chaves, essas chaves vão fornecer a ordem pela qual as linhas vão ser agrupadas. No caso da imagem anterior, a chave de partição é a coluna "*user_id*" e a chave fornecida para estabelecer a ordem de aglomeração é a coluna "*tweet_id*".

Clustered by tweet_id

gmason	[1765, author]: phenry	[1765, body]: Give me liberty or give me death	[1742, author]: gwashington	[1742, body]: I chopped down the cherry tree
ahamilton	[1797, author]: jadams	[1797, body]: A government of laws not men	[1742, author]: gwashington	[1742, body]: I chopped down the cherry tree

Figura 33 – O mesmo exemplo de colunas compostas (Disposição física). Retirado de (DataStax, 2012)

Na figura anterior pode ser observado como são armazenadas as colunas compostas a um nível físico. Como a chave de partição definida é o valor da coluna "*user_id*", foram criadas duas linhas correspondentes aos diferentes utilizadores existentes. No armazenamento físico de colunas compostas a chave de partição assume também o papel de chave de linha, o

equivalente a uma chave primária no sistema relacional. Também pode ser observado na figura anterior que o valor da chave utilizada para definir a ordem de aglomeração, está presente no nome das colunas. O ponto mais relevante a salientar é o facto de as 4 linhas presentes na Figura 32 serem representadas a um nível físico, como apenas duas linhas.

Colunas que Expiram: Tradicionalmente as bases de dados recorrem a excertos de código externos para eliminarem dados que precisam de expirar. Em *Cassandra*, todavia, é possível definir o tempo de vida de uma coluna aquando do processo de inserção de novos dados.

Colunas Contador: Este tipo de colunas são utilizadas para armazenar um número que incrementalmente conta a ocorrência de um determinado evento. Estas colunas podem, por exemplos, ser utilizadas para armazenar o número de visitas de utilizadores a um determinado endereço *web*. Apesar de parecer um conceito bastante simples, a sua implementação é bastante complexa. O sistema *Cassandra* foi desenhado para operar num ambiente distribuído, por consequência um contador pode estar a ser atualizado em 2 nós diferentes simultaneamente, enquanto um terceiro nó está a tentar aceder ao valor desse contador. Vários processos têm de ser levados a cabo para que o terceiro nó aceda à correta versão desse contador.

6.2.4 A Estrutura de um armazém de dados em Cassandra

Do estudo efetuado nos pontos anteriores, foi possível obter várias informações que possibilitaram desenhar um esquema de um armazém de dados para o sistema Cassandra. A primeira conclusão a que se chegou foi que a estrutura do armazém de dados relacional teria de ser completamente desnormalizada. A ideia de que cada linha no sistema Cassandra irá possuir 180 colunas, não é fácil de aceitar pelo instinto normalizador que todos os engenheiros informáticos foram desenvolvendo ao longo da sua vida académica. Segundo (Sharma, 2014) o sistema *Cassandra* consegue armazenar linhas compostas por dois milhares de milhão de colunas. Considerando essa perspetiva, as 180 colunas do armazém de dados não representam qualquer problema.

Do subponto anterior ainda se chega à conclusão de que, para sustentar o armazém de dados em causa, será necessário recorrer a famílias de colunas fixas. No entanto, criou-se uma dúvida: teria a utilização de colunas compostas um impacto positivo ou negativo na execução de consultas? Uma outra dúvida encontra-se dentro dessa dúvida… Ao optar pela utilização de colunas compostas, quais seriam os elementos a ser fornecidos como chave de partição e restantes chaves para definir a ordem da agregação?

Os Esquemas *Cassandra* são definidos para responder a *queries* específicas mas, para se representar a estrutura de um armazém de dados, é necessário criar um esquema genérico, capaz de responder a várias consultas. A solução perfeita para o desenho do esquema do armazém de dados em Cassandra, seria simplesmente criar 4 esquemas diferentes, cada um orientado para as 4 consultas definidas na secção 4.4.1. No entanto, essa solução não permitiria

avaliar a viabilidade do sistema *Cassandra* como suporte a um armazém de dados do mundo real, onde um único esquema tem de conseguir responder a várias consultas diferentes.

Os exemplos de colunas compostas consultados apenas são aplicáveis a pequenos exemplos, não existe nenhuma documentação ou informação capaz de informar sobre o melhor método para se moldar 180 colunas compostas distribuídas por 7 dimensões diferentes. Também não existem artigos académicos ou empresariais, de como moldar um esquema de um armazém de dados no sistema *Cassandra*. Como aconteceu noutras etapas do processo de elaboração desta dissertação, não foi possível virar a cara a conhecimento que surgiu de forma eminente. Assim, decidiu-se desenhar 3 esquemas diferentes mas apenas um deles poderia ser utilizado na execução dos testes de performance. Para decidir qual o selecionado, um pequeno teste comparativo teve de ser levado a cabo. Os 3 esquemas desenhados são os seguintes:

6.2.4.1 Esquema 1

O esquema 1 é o esquema mais simples que se pode apresentar. O esquema é representado por apenas 180 colunas padrão, atribuídas a cada linha da tabela de factos do armazém de dados relacional. Se existe um esquema capaz de fornecer expressividade suficiente para responder a todos os tipos de consultas, este será esse esquema. No entanto, a utilização de simples colunas padrão, poderá significar que não se está a tirar total partido das capacidades do sistema *Cassandra*, o que pode representar uma ligeira desvantagem de performance em relação a outros sistemas competidores.

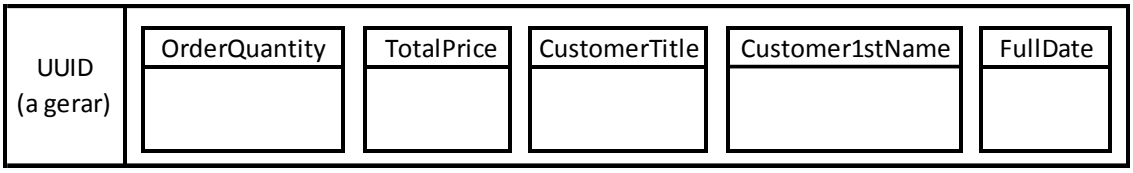

Figura 34 – 1º esquema de um armazém de dados no sistema *Cassandra* (**Disposição Física**)

Na figura anterior está representada uma estrutura parcial do que constituirá uma linha do primeiro esquema desenhado. Apenas estão representadas duas medidas e alguns elementos das dimensões. Esta é só a ilustração de uma estrutura parcial, a ilustração total teria de conter 180 colunas, o que não seria plausível de ilustrar neste documento. É de salientar que a chave de fila será representada por um *UUID*[14] gerado pela aplicação aquando do processo de inserção de dados.

6.2.4.2 Esquema 2

O segundo esquema criado surge do facto de que, no caso da utilização de colunas compostas, as chaves fornecidas depois da chave de partição (mas que não lhe pertencem) estabelecem a ordem pela qual as linhas são aglomeradas. Nessa situação o valor dessas chaves irá estar (a um nível físico) presente no nome da coluna. Partindo desse princípio decidiu-se criar um esquema, com recurso a colunas compostas, em que a chave de partição é um *UUID* e as chaves

[14] Um Identificador único universal (***U****niversally* ***U****nique* ***Id****entifier*) é um standard de identificação utilizado na construção de software. A sua utilização tem por objetivo permitir que sistemas distribuídos consigam identificar unicamente informação, sem ser necessário uma significante coordenação central.

adicionais são as medidas presentes na tabela de factos. O que este esquema implica em primeiro plano é que nenhumas linhas serão aglomeradas, visto que a chave de partição é um *UUID*, que será sempre único. Mas, por outro lado, as medidas irão estar todas presentes no nome de cada coluna, o que poderá implicar uma redução no tempo necessário para calcular a aglomeração de uma medida baseada num atributo específico.

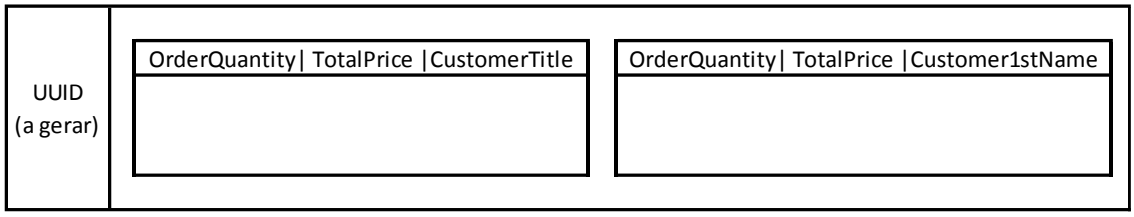

Figura 35 - 2º esquema de um armazém de dados no sistema *Cassandra* (**Disposição Física**)

Este esquema representa apenas uma teoria. A utilização deste esquema pode vir a resultar numa degradação da performance das consultas, contrariando completamente o objetivo pretendido. É de salientar mais uma vez que a figura representa apenas uma estrutura parcial. No esquema real, estarão presentes a um nível físico 11 medidas no nome de cada uma das 169 colunas restantes.

6.2.4.3 Esquema 3

A criação de um terceiro esquema deve-se à tentativa de utilizar as colunas compostas do sistema Cassandra para aquela que é uma das suas utilidades mais pertinentes, as séries temporais (Sharma, 2014).

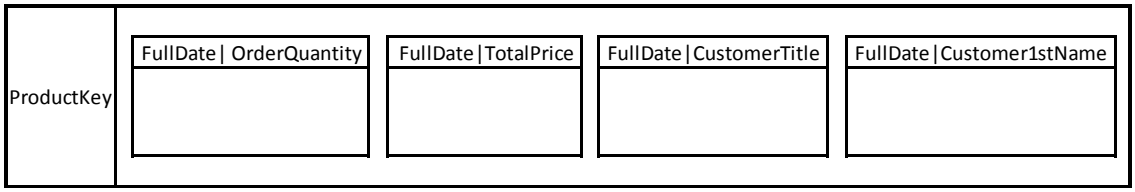

Figura 36 - 3º esquema de um armazém de dados no sistema Cassandra (**Disposição Física**)

Neste esquema a chave de partição é representada pela chave de cada produto existente. Assim, todas as linhas de vendas de um determinado produto irão ser aglomeradas numa única linha. Como a chave fornecida juntamente com a chave de partição é a o atributo relativo à data, as linhas serão ordenadas segundo uma sequência temporal. A utilização dos produtos como chave de partição parte apenas da presunção de que os produtos vendidos são o ponto mais importante do armazém de dados a criar. Este esquema, também se apresenta como uma teoria.

6.3 Programação e Migração de dados

Exatamente como no caso do sistema *MongoDB*, durante o estudo realizado não se encontrou nenhuma ferramenta livre capaz de migrar os dados relacionais para os esquemas *Cassandra* desenhados. Foi necessário assim recorrer à programação para se conseguir proceder á migração dos dados. Também como no caso do sistema *MongoDB*, um driver é fornecido para que se possa programar interações com o sistema. Para isto ser possível é necessário, mais uma vez, realizar a transferência do *driver C#* através do gestor de pacotes *Nuget*. É possível proceder à instalação do pacote percorrendo manualmente os repositórios ou simplesmente executando a seguinte linha na consola *Nuget*: `Install-Package CassandraCSharpDriver`.

Mais uma vez, é extremamente importantes perceber atempadamente quais são os tipos de dados suportados pelo sistema de destino. Ao contrário do que acontece no sistema MongoDB, no sistema Cassandra o driver não irá automaticamente converter os dados de uma variável ou objeto para o formato equivalente no novo sistema. Em *Cassandra* os dados são inseridos através de um método igual ao das bases de dados *SQL*, ou seja, através de uma *string* começada pela palavra "*INSERT*". Esta situação implica que os valores a inserir sejam formatados previamente para que o sistema, ao ler essa *string*, consiga detetar os diferentes tipos de dados.

Tabela 9 – Equivalência entre os tipos de dados do sistema Cassandra e das linguagens *.NET*

Tipos de dados *CQL3*	Tipos de dados *.NET*
bigInt, counter	*long*
bolean	*bool*
decimal, float	*float*
double	*double*
int	*int*
uuid, timeuuid	*System.Guid*
text, varchar	*string (UTF8)*
timestamp	*System.DateTimeOffset*
variant	*System.Numeric.BigInteger*

Mais uma vez por contraposto ao sistema *MongoDB*, o sistema Cassandra obriga a que seja criado um esquema para cada família de documentos, antes de as colunas serem inseridos no sistema. Assim foi necessário utilizar a linguagem *C#* para criar os 3 esquemas referidos anteriormente.

Os excertos de código a serem apresentados têm por base as informações disponibilizados pela *DataStax* sobre o *driver C#* (DataStax, 2014b) e sobre a linguagem *CQL3* (DataStax, 2014c). Mais uma vez, devido ao grande número de atributos, não será possível apresentar o código completo da solução criada. Deste modo, serão aqui apresentados exemplos de código que recorrem apenas a alguns elementos presentes no armazém de dados. Seguindo a filosofia empregue anteriormente, por uma questão de simplicidade e para não tornar o documento

extenso e de fastidiosa leitura, apenas serão apresentadas as linhas de código mais relevantes. Partindo do princípio que o leitor conseguirá enquadra-las sem problema na estrutura, corretamente definida, de uma aplicação C#.

```
using Cassandra; // Referenciar os DLLs instalados com o pacote Nuget
// Variável que irá armazenar a referência da ligação a um nó do sistema
private Cluster cluster;
// Variável que irá armazenar a referência para a sessão a iniciar
private ISession sessao;

// Adicionar um nó, e criar uma instância "Cluster"
cluster = Cluster.Builder().AddContactPoint("127.0.0.1").Build();
// Recolher informações sobre o "Cluster" em causa
Metadata metadata = cluster.Metadata;
//Estabelecer uma ligação entre a aplicação e o Cluster
sessao = cluster.Connect();
```

Código 16 – Excerto de código *C#* necessário para efetuar a ligação a um sistema *Cassandra*

No excerto anterior, estão ilustradas as linhas de código necessárias para se estabelecer uma ligação a um conjunto de nós (*Cluster*) do sistema Cassandra. No caso ilustrado, apenas um nó é adicionado a esse conjunto. No sistema *MongoDB* optou-se pela utilização de um interface visual, para se criar uma base de dados. Neste sistema optou-se por utilizar código *C#* para realizar o processo equivalente, ou seja, criar um *KeySpace*.

```
sessao.Execute("CREATE KEYSPACE IF NOT EXISTS Dissertacao1  WITH
     replication = {'class': 'SimpleStrategy', 'replication_factor' : 1}");

cluster.Shutdown(); //Terminar a ligação
```

Código 17 - Excerto de código *C#* para a criação de um *KeySace* em *Cassandra*

Na criação de um *KeySpace* é necessário definir qual é a estratégia de replicação a utilizar. Existem dois tipos de estratégia: "*NetworkTopologyStrategy*", utilizada quando se pretende ter um sistema a funcionar sobre diferentes conjuntos de nós, que podem ou não estar fisicamente em locais distintos. "*SimpleStrategy*", utilizada quando apenas se pretende utilizar um único grupo de nós, usualmente presentes na mesma localização física. É também necessário definir qual o fator de replicação a utilizar, este irá definir quantas cópias da mesma linha irão existir num conjunto de nós, sendo que cada réplica é armazenada num nó diferente.

```
sessao.Execute("CREATE TABLE IF NOT EXISTS
   Dissertacao.FactsDimsCassandra( id uuid PRIMARY KEY, OrderQuantity int,
   totalPrice, CustomerTitle text, CustomerFirstName text, ProductKey int,
   FullDateAlternateKey timestamp)");
cluster.Shutdown(); //Terminar a ligação
```

Código 18 – Excerto de código *C#*, ilustrativo da criação do esquema 1

Dos 3 esquemas a criar, o esquema 1 é o único que não possui colunas compostas. Desse modo a chave de fila (*Row Key*) é simplesmente definida através da utilização das palavras "*PRIMARY KEY*" logo após o atributo referente ao tipo de dados dessa coluna.

```
sessao.Execute("CREATE TABLE IF NOT EXISTS
  Dissertacao.FactsDimsCassandra2( id uuid, OrderQuantity int,
  totalPrice, CustomerTitle text, CustomerFirstName text, ProductKey int,
  FullDateAlternateKey timestamp,
  PRIMARY KEY( id, UnitPrice, UnitPriceDiscount,
  LineTotal, TaxAmount, Freight, TotalPrice, TotalPriceWithoutTax,
  TotalPriceWithoutFreight, TotalPriceWithoutDiscount,
  TotalPriceWithoutTaxFreightDiscount))");
cluster.Shutdown(); //Terminar a ligação
```

Código 19 – Excerto de código *C#,* ilustrativo da criação do esquema 2

```
sessao.Execute("CREATE TABLE IF NOT EXISTS
  Dissertacao.FactsDimsCassandra3( id uuid, OrderQuantity int,
  totalPrice, CustomerTitle text, CustomerFirstName text, ProductKey int,
  FullDateAlternateKey timestamp,
  PRIMARY KEY(ProductKey, FullDateAlternateKey))");
cluster.Shutdown(); //Terminar a ligação
```

Código 20 - Excerto de código *C#,* ilustrativo da criação do esquema 3

Os dois excertos de código anteriores ilustram o código necessário para criar os 2 esquemas restantes. Estes excertos só diferem do código apresentado para o esquema 1 na definição da chave de fila. Nestes dois exemplos, como é necessário criar colunas compostas, as palavras "*PRIMARY KEY*" deixam de ser utilizadas imediatamente após a definição de uma coluna, e passam a ser utilizada no fim. O primeiro atributo presente dentro da "*PRIMARY KEY*" é a chave de partição, todos os outros atributos seguintes irão definir a ordem pela qual as linhas serão aglomeradas.

```
sessao.Execute("INSERT INTO
  Dissertacao.FactsDimsCassandra1( id, OrderQuantity, totalPrice,
  CustomerTitle, CustomerFirstName, ProductKey, FullDateAlternateKey)
  VALUES("+ System.Guid.NewGuid().ToString() +", 15, 350.23, null,
  'Daniel', '' +DateTime.Now.ToString("yyyy-MM-dd") +'')");
cluster.Shutdown(); //Terminar a ligação
```

Código 21 – Excerto de código *C#* responsável por inserir dados nos 3 esquemas

Este último excerto de código limita-se a inserir dados em qualquer um dos 3 esquemas criados. A única mudança a efetuar é no nome da família de colunas a utilizar. Este código apresenta um exemplo bastante simples. Na migração de dados do armazém *SQL Server* para o sistema Cassandra, algumas conversões de dados tiveram de ser realizadas.

Como na execução de *queries* o *driver Cassandra* se limita a interpretar uma *string*, todos os dados oriundos da base de dados relacional têm de ser convertidos para texto, de modo a serem concatenados com a *string* que representa o código da *querie*. Os únicos problemas encontrados foram o facto de a linguagem *CQL3* não entender a vírgula (,) como o separador numérico dos algarismos de números não inteiros. A utilização do ponto (.) como separador numérico teve de ser forçada pelo código aquando da conversão para *string*. Outra questão surgiu na conversão de datas; para que o sistema interprete corretamente o valor, esta tem que ser apresentada no formato Ano-Mês-Dia. Por fim, tal como nas consultas *SQL*, todos as "palavras" têm de ser colocadas entre pelicas ('). Como o armazém de dados relacional foi

preenchido com dados na língua inglesa, os dados a migrar possuem bastantes *strings* com pelicas individuais que precisam de ser "escapadas", o contrário irá resultar na falha da *query* a ser executada.

6.4 Consultas em Cassandra

Como pode ser observado nos exemplos de código anteriores, a linguagem *CQL3* utilizada pelo sistema Cassandra é bastante parecida com a linguagem *SQL* a todos os níveis.

```
cqlsh:demodb> SELECT * FROM emp where empID IN (130,104) ORDER BY deptID ASC;

 empid | deptid | first_name | last_name

-------+--------+------------+-----------
   130 |      5 |     sughit |     singh
   104 |     15 |       jane |     smith
```

Figura 37 – Exemplo de consulta utilizando a linguagem CQL3. Retirado de (Apache, 2014c)

A última afirmação também é válida no caso das consultas. A figura anterior é um exemplo da estrutura de uma consulta em *CQL3* e é rapidamente observável que a linha apresentada é indistinguível de uma consulta realizada num sistema relacional. Nesta fase do processo de investigação começou-se a estudar como é que se construiriam as consultas de performance estabelecidas, na linguagem *CQL3*. Algo que não deveria ser complicado, visto esta linguagem ser bastante parecida com a tradicional linguagem *SQL*. Infelizmente, existe uma pequena curiosidade do sistema Cassandra que não era conhecida. Este sistema não suporta funções de agregação, para além de contar (*COUNT)*. Esta situação implica que funções que permitam calcular a média (*AVG*), soma (*SUM*), mínimo (*MIN*) e máximo (*MAX*) não são suportadas pela linguagem *CQL3*, ou outra linguagem do sistema. Para conseguir realizar estas operações seria necessário recorrer a programação ou à utilização de funções *Hadoop MapReduce*, o que derrotaria o objetivo de as consultas serem executadas numa só execução, sem ser necessário que os resultados sejam tratados na lógica de uma aplicação.

6.4.1 Resolução do problema - Alternativa 1: Hive

6.4.1.1 Hive (Apache, 2014b)

À medida de que a rede social F*acebook* foi crescendo, as necessidades de armazenamento e de processamento de dados foram aumentando. Inicialmente todos os armazéns de dados estavam implementados sobre instâncias *Oracle*, mas a explosão no volume de dados começou a causar problemas de instabilidade no sistema (White, 2012). Outras opções teriam inevitavelmente que ser consideradas. A migração para *Hadoop* aparentava ser a melhor opção, mas a falta de suporte para a construção de armazéns de dados e a dificuldade de utilização de *MapReduce* para os veteranos da linguagem *SQL* revelava-se um problema.

Deste modo, o *Facebook* optou por desenvolver uma solução para este problema internamente. Surge então o sistema *Hive*, apresentado inicialmente nos artigos (Thusoo et al., 2009, Thusoo et al., 2010). Este sistema representa uma infraestrutura que permite a construção de armazéns de dados, "em cima" da *framework Hadoop*, e do seu sistema de armazenamento, o *HDFS (Tiwari, 2011)*. Este sistema, no entanto, não é considerado como uma base de dados no verdadeiro sentido da palavra. As restrições de desenho do *Hadoop* e do HDFS impõe limites naquilo que o *Hive* pode fazer, sendo uma das suas mais flagrantes falhas não disponibilizar métodos para se atualizarem ou eliminarem registos individuais (Capriolo et al., 2012). A importância deste projeto pode ver-se pelo número de organizações e programadores que já contribuíram para o projeto apache *Hive* depois de este ter sido disponibilizado como fonte aberta. Segundo (Huai et al., 2014), mais de 100 equipas já efetuaram esforços técnicos que contribuíram para a correção de mais de 3000 problemas. E o elevado ritmo de desenvolvimento continua.

A utilização das caraterísticas de "Armazém de dados" do sistema *Hive* nunca foram consideradas na elaboração desta investigação. Este sistema não é considerado uma base de dados NoSQL, e muitas fontes não o consideram, sequer, como uma base de dados "completa". Neste ponto essas caraterísticas continuam a não ser o foco do interesse. Do ponto de vista do problema a resolver, o ponto mais interessante deste sistema é a linguagem utilizada para a realização de análises, conhecida como *HiveQL* (Hive Query Language). A linguagem *HiveQL* permite que análises profundas e complexas sejam levadas a cabo através de uma sintaxe bastante idêntica à da linguagem SQL. Esta abordagem permite diminuir a diferença cultural para programadores SQL, e ao mesmo tempo permite que armazéns de dados relacionais possam ser migrados para uma arquitetura *Big Data*, que lhes permitirá escalar muito mais facilmente (Barbierato et al., 2013).

O sistema *Cassandra* disponibiliza nativamente métodos que permitem ligar o seu armazenamento à *framework Hadoop*, para que funções *MapReduce* sejam executadas sobre os dados. As consultas na linguagem *HiveQL* são, no momento da execução, convertidas em funções *MapReduce*. O lado positivo deste fator é que torna-se possível executar consultas *HiveQL* nos dados armazenados no sistema Cassandra. O lado negativo deste fator, é que as consultas *MapReduce*, não são indicadas para o processamento de análises em tempo real (Barbierato et al., 2013) As despesas da gestão e organização de processos *Map* e *Reduce* originam um elevado valor de latência (Tiwari, 2011). O que resulta em consultas demoradas, mesmo no caso de um pequeno volume de dados. A linguagem *HiveQL* estará no máximo da sua força, no caso de grandes volumes de dados, onde a sua natureza *MapReduce* a torna bastante rápida em comparação com outros sistemas (Tiwari, 2011). No entanto, outro problema surge. Infelizmente o sistema *Hive* não é executável na plataforma *Windows*.

6.4.2 Resolução do problema – Alternativa 2: Emulador HDInsights

A solução para este problema está na plataforma de computação móvel *Microsoft Azure* (Microsoft, 2014d), mais especificamente no serviço *HDInsights (Microsoft, 2014c).* Este serviço

representa a implementação de uma solução *Big Data* que possui como base o sistema Apache *Hadoop*. A *Microsoft* em grande parte limitou-se a utilizar a distribuição de fonte aberta do *Hadoop* e adicionou-lhe as funcionalidades necessárias para o sistema ser compatível com plataformas Windows. Todos os outros componentes mantem-se praticamente inalterados. Deste modo garantindo que tudo funciona corretamente, e que não existem conflitos entre diferentes partes do sistema (Sarkar, 2014). A Microsoft também tornou bastante fácil ligar os sistemas *Hadoop* alocados na nuvem às poderosas ferramentas de *Business Intelligente* do seu catálogo. É assim bastante simples gerar análises profundas e relatórios interativos, tudo sobre um mesmo teto (Sarkar, 2014). O sistema *HDInsight* fornece acesso a vários serviços *Hadoop*, sendo um deles o sistema *Hive*.

Tentou-se utilizar o serviço *HDinsight*, para resolver o problema "*Hive em windows*". A *Microsoft* disponibiliza 150€ de utilização gratuita do sistema para efeitos de teste. Apesar de o demo disponibilizado ser mais que suficiente para a utilização requerida, a requisição obrigatória de número de telefone e cartão de crédito aquando do processo de registo, parecem bastante rígidas. Felizmente é disponibilizado um emulador do sistema *HDInsight* (Microsoft, 2014b) para que programadores possam criar e testar aplicações desenvolvidas para interagir com o sistema real (Jorgensen et al., 2014). Este emulador foi instalado na máquina local e alguns testes foram realizados de imediato.

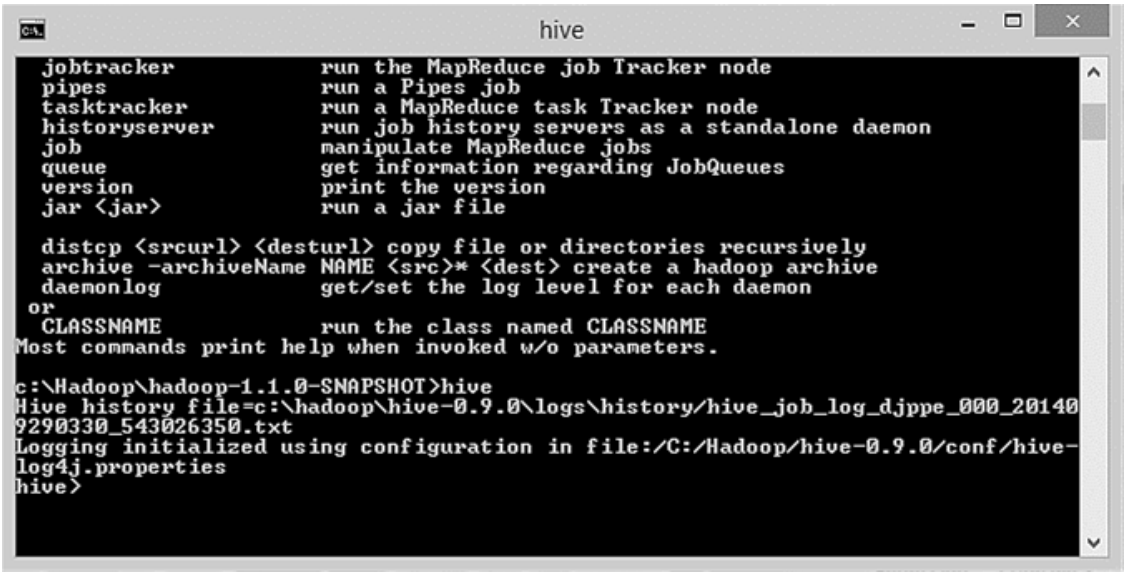

Figura 38 - Consola *Hive* do Emular *HDInsights* a ser executado na máquina local

Este passo acabou por se revelar numa precipitação. Um estudo subsequente levou à conclusão de que é realmente possível integrar a *framework Hadoop* no sistema Cassandra, apesar de várias configurações complexas serem necessárias, como pode ser observado em (Nicol, 2013). O primeiro problema nesta abordagem surge no facto de que, para se integrar Hadoop e *Cassandra*, os ficheiros e configurações do sistema *Hadoop* têm de estar presentes em todos os nós deste último. Neste caso específico, a *framework Hadoop* teria de correr no emulador ou na nuvem e, consequentemente nunca seria possível executá-los no mesmo nó que o sistema *Cassandra*. O segundo problema desta abordagem é o facto de o sistema *Hive*

necessitar de configurações especiais para conseguir compreender os dados e a estrutura de armazenamento do sistema *Cassandra*. Após um estudo profundo, verificou-se que só existe uma solução exequível em tempo útil para a resolução deste problema…

6.4.3 Resolução do problema – Alternativa 3: DataStax Enterprise

A única solução já existente para este problema é a utilização do sistema *DataStax Enterprise* (DataStax, 2014d). Este sistema representa a versão comercial do sistema instalado anteriormente. Este possui igualmente um sistema de armazenamento *Cassandra*, mas disponibiliza muitas outras funcionalidades, como a inclusão direta no sistema da *framework Hadoop* de onde se salienta o sistema *Hive*. Esta solução da *DataStax* é de utilização livre para programadores e estudantes que pretendam avaliar e aprender a desenhar aplicações para este sistema. Este sistema, no entanto, só pode ser executado em plataformas Unix.

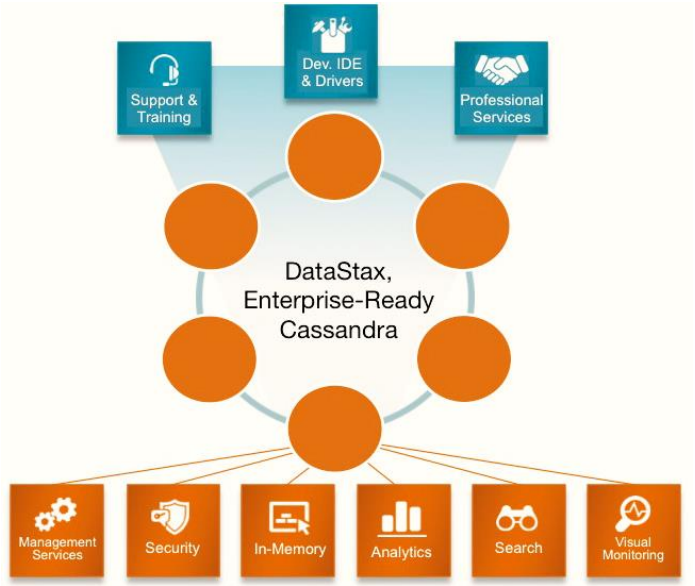

Figura 39 – Caraterística do sistema *DataStax Enterpsie*. Retirado de (DataStax, 2014d)

De modo a conseguir alcançar o objetivo de executar as consultas especificadas no sistema Cassandra, decidiu-se abdicar de dois objetivos secundários estabelecidos anteriormente: realizar todos os testes e migração de dados sobre sistemas instalados numa plataforma *Windows* e toda a programação ser levada a cabo na linguagem *C#.*

Assim procedeu-se à instalação na máquina local da última distribuição *Ubuntu* disponível. É bastante simples utilizar este sistema, e realizar as consultas na linguagem *Hive*. Como o sistema *Ubuntu* consegue aceder aos ficheiros do sistema Windows, não é necessário migrar os dados *Cassandra* para um outro sistema operativo, basta apenas modificar o caminho no ficheiro de configuração do sistema Cassandra (*Ubuntu*). A inicialização do sistema *Cassandra* e posteriormente do sistema *Hive* apenas requer a execução de 2 comandos. A programação orientada para a execução de consultas no sistema *Hive* foi levada a cabo através da linguagem Java e do respetivo driver (já disponibilizado com o sistema).

6.4.3.1 Seleção do esquema Cassandra a utilizar

```
SELECT productName, AVG(lineTotal)
FROM factsDimsCassandra1
GROUP BY productName
```

Código 22 – Consulta *HiveQL* elaborada para seleção do esquema *Cassandra*

A única tarefa restante antes de se proceder à realização das consultas de performance é aferir qual dos 3 esquemas criados, apresentará a melhor performance. Para alcançar esse efeito carregaram-se 150 mil linhas para cada um dos 3 esquemas *Cassandra* e criou-se uma simples consulta em *HiveQL*. A ideia desta consulta era possuir uma estrutura simples contendo uma única função de agregação.

Tabela 10 – Resultados da Exequção da querie HiveQL nos 3 esquemas Cassandra criados

	Tempo de Execução (segundos)	Obteve-se o resultado esperado?
Esquema 1	1113,176 (18 Min 33 Seg)	✓
Esquema 2	1059,271 (17 Min 39 Seg)	✖ Não se obteve resultado
Esquema 3	1089,059 (18 Min 09 Seg)	✖ Diferente do resultado esperado

Antes de se proceder à execução das consultas, realizou-se exatamente o mesmo processo no armazém de dados *SQL Server*. Para além de validar os resultados obtidos, este passo tinha como objetivo avaliar se a utilização de colunas compostas teria alguma influência nos resultados. Os resultados obtidos confirmaram os receios inicias. A linguagem *HiveQL* não está preparada para interpretar corretamente a estrutura de colunas compostas do sistema *Cassandra*. No esquema 2 não se obteve resposta, apesar de a consulta ser executada e ter a sua execução concluída pelo sistema. Muito provavelmente, esta situação deve-se ao facto de o atributo "lineTotal" estar presente no cabeçalho das colunas, não estando a linguagem *HiveQL*, preparada para apresentar o resultado da média obtida. No caso do esquema 3, obteve-se o resultado esperado em relação ao atributo "*productName*", mas a média do atributo "*lineTotal*" apresentada para cada um dos produtos era inferior aos números esperados. Assim o esquema 1, o mais simples e trivial de todos será o utilizado. Este pequeno teste serviu para avaliar outro receio, o tempo das consultas *Hive*/*MapReduce*. Cerca de 18 minutos para um volume de 150 mil linhas é um valor proibitivo. De salientar que os resultados apresentados representam o valor médio de duas repetições da referida consulta, em cada um dos esquemas. Consultas a realizar: Linguagem HiveQL

- **Consulta A:**

```
SELECT SUM(unitpricediscount*orderquantity) AS TotalDescontos
FROM factsdimscassandra
WHERE calendaryear=2008 AND orderquantity BETWEEN 2 AND 10
```

Código 23 – Consulta A representada na linguagem *HiveQL*

- **Consulta B:**

```
SELECT productname, calendaryear, SUM(freight) AS Total
FROM factsdimscassandra
WHERE productcategoryname='Accessories' AND
       shipmethodname ='CARGO TRANSPORT 5'
GROUP BY calendaryear, productname
ORDER BY calendaryear, productname
```

Código 24 – Consulta B representada na linguagem *HiveQL*

- **Consulta C:**

```
SELECT territoryname, productcategoryname, AVG(totalprice) AS media
FROM  factsdimscassandra
WHERE customercountryregionname!='United States' AND productcolor='Red'
      AND employeegender='F' AND fulldatealternatekey BETWEEN
      unix_timestamp('2010/06/01', 'yyyy-MM-dd') AND
      unix_timestamp('2012/06/01', 'yyyy-MM-dd')
GROUP BY territoryname, productcategoryname
ORDER BY territoryname, media
```

Código 25 – Consulta C representada na linguagem *HiveQL*

- **Consulta D:**

```
SELECT productname, territoryname, calendaryear, AVG(employeesalesquota)
       AS Quota, SUM(((unitprice-unitpricediscount)*orderquantity)) AS
       semDesc, AVG(productvendorstandardprice) AS MediaPrecoVendedor
FROM  factsdimscassandra
WHERE territorycountryregionname!='Australia' AND shipmethodname!='CARGO
      TRANSPORT 5' AND fulldatealternatekey BETWEEN
      unix_timestamp('2005/03/01', 'yyyy-MM-dd') AND
      unix_timestamp('2014/03/01', 'yyyy-MM-dd')
GROUP BY productname, territoryname, calendaryear
ORDER BY productname, territoryname
```

Código 26 – Consulta D representada na linguagem *HiveQL*

6.5 Conclusões

Todos os processos de criação do armazém de dados no sistema *Cassandra* necessitaram de estudos mais aprofundados do que aconteceu no sistema *MongoDB*. De facto, todo o processo de migração de dados e desenho de consultas foi mais complexo neste sistema. Esta situação deve-se em grande parte ao elevado tempo que foi necessário para compreender a sua "diferente" estrutura de armazenamento.

Este capítulo finaliza numa situação ligeiramente agridoce. Um elevado tempo e alguma persistência foram necessários para se conseguir contornar corretamente os vários problemas encontrados. Algo que se alcançou com sucesso. No entanto, as perspetivas para os resultados dos testes de performance não são nada animadoras. As consultas realizadas com o intuito de selecionar o esquema a utilizar revelam valores bastante elevados.

Não obstante todas as situações descritas, o objetivo de criar e povoar um armazém de dados no sistema Cassandra foi cumprido com sucesso. As consultas criadas também foram adaptadas com sucesso ao sistema em causa. Apesar da grande travessia necessária para alcançar o sucesso nesse aspeto.

7 Testes de Performance

Neste capítulo serão realizados testes de performance às 3 bases de dados selecionadas. Estes testes irão incidir sobre a velocidade de resposta às quatro consultas criadas anteriormente. Para além dos testes, também serão elucidados quais os métodos e ambientes em que todos os processos se desenrolaram. É esperado que este capítulo represente o culminar de todas as investigações realizadas até esta fase da dissertação.

7.1 Regras e ambiente de realização dos testes

As especificações de *hardware* da máquina onde foram realizados os testes podem ser encontradas no Anexo F deste documento. É muito importante salientar que os testes foram realizados numa máquina com alguns anos. Esta situação representa desde logo uma desvantagem para qualquer tipo de base de dados. A grande maioria das organizações utiliza nos seus armazéns de dados discos rígidos SSD e grandes quantidades de memória *RAM* bastante rápida. A máquina em causa apresenta um disco rígido com mais de quinze mil horas de utilização e uma memória *RAM* curta e lenta (pelos padrões atuais). Não há dúvida que estes fatores irão ter uma influência sobre os resultados. É esperado que os resultados a obter sejam ligeiramente desenquadrados da realidade informática atual.

Visto por um outro ângulo, estas especificações de *hardware* vão garantir uma verdadeira análise sobre as capacidades dos sistemas selecionados. As pequenas diferenças entre os vários sistemas vão ser potenciadas a uma outra escala, sendo que pequenas diferenças de performance irão ter um impacto muito maior, afetando diretamente o tempo de resposta a uma consulta. É fundamental referir que este não é um teste direto entre os três sistemas selecionados. Não é só a base de dados que está em causa, mas também o esquema, o tipo de armazenamento e o método de realização de consultas.

Para se obter um maior conhecimento sobre as capacidades de cada sistema, decidiu-se observar o tempo de resposta a consultas realizadas sobre diferentes volumes de dados. Assim estabeleceram-se 6 diferentes cargas de dados: 250 mil linhas, 750 mil linhas, 1 milhão de linhas, 2 milhões de linhas, 3 milhões de linhas e 5 milhões de linhas. No caso do sistema *SQL Server*, apenas a tabela de factos é que vê o seu volume de dados mudar. Todas as dimensões deste sistema mantêm o seu volume, em todas as consultas.

Os resultados apresentados no subponto seguinte representam o valor médio obtido com dez execuções distintas de cada uma das quatro consultas, para cada um dos seis volumes de dados definidos. No sistema *Cassandra*, apenas foram realizadas duas execuções de cada consulta. Os motivos que originaram esta situação serão elucidados posteriormente. Todos os sistemas utilizados possuem os dados armazenados na mesma partição do disco rígido, sendo que a cache de cada sistema foi limpa entre cada consulta.

7.2 Resultados dos testes de performance: *SQL Server* 2014

Os testes ilustrados neste subponto foram realizados no sistema *Microsoft SQL Server* 2014, versão 12.0.2254 *Enterprise Edition*. Esta base de dados encontrava-se a ser executada numa máquina com sistema operativo *Microsoft Windows* 8.1 *Pro*, versão 6.3 (*Build* 9600). Os testes neste sistema foram executados através do programa de administração "*SQL Server Management Studio*". Testes preliminares revelaram que esta solução apresentava resultados ligeiramente melhores (mas quase indistinguíveis), em relação a consultas executadas e cronometradas dentro de uma aplicação *C#*.

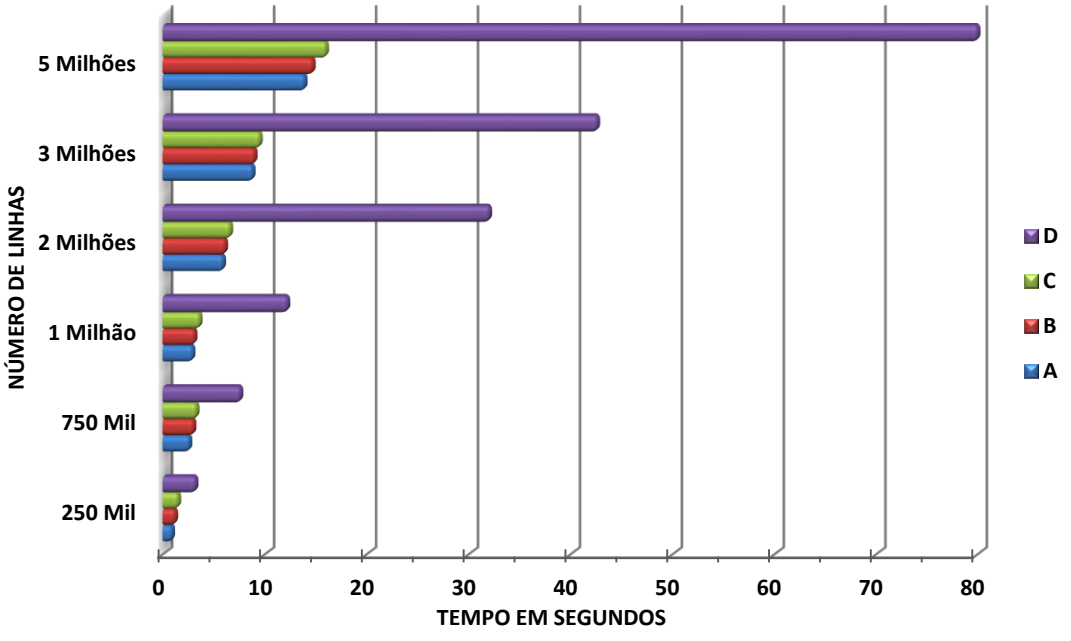

Figura 40 – Gráfico ilustrativo do tempo de execução de cada consulta

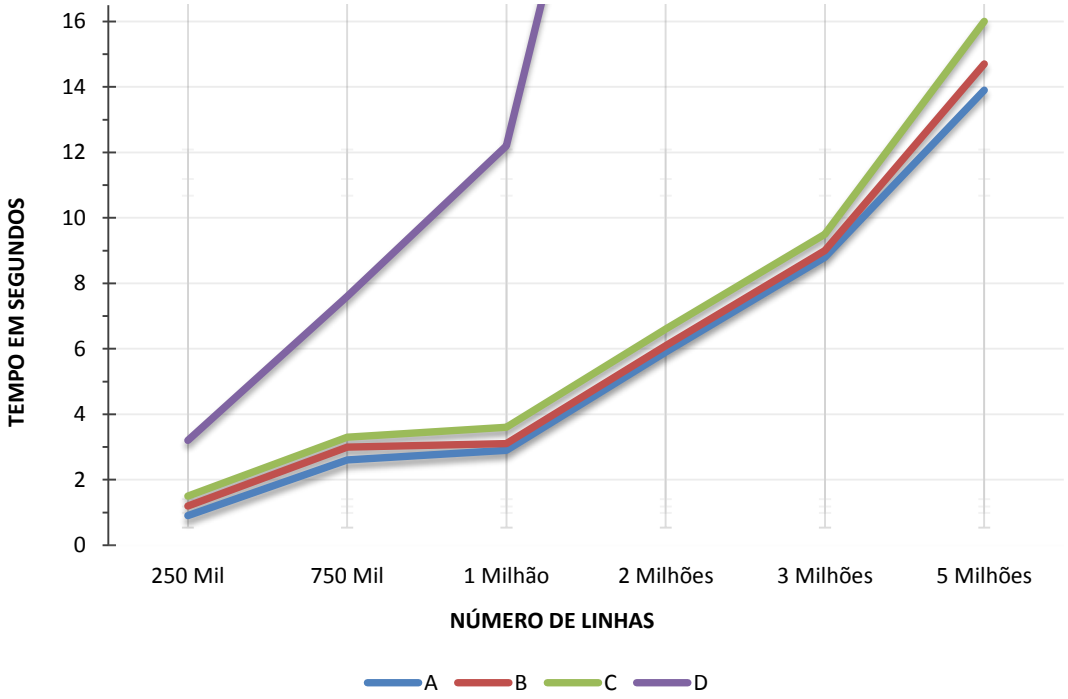

Figura 41 – Gráfico ilustrativo da evolução do tempo de resposta com o aumento de volume de dados

Tabela 11 - Tempo de resposta (em milissegundos) do sistema SQL Server a cada consulta

	Consulta A	Consulta B	Consulta C	Consulta D
250 Mil	869	1235	1487	3158
750 Mil	2591	2956	3322	7607
1 Milhão	2857	3110	3580	12215
2 Milhões	5853	6101	6617	32039
3 Milhões	8787	9011	9489	42575
5 Milhões	13856	14740	15976	80511

Não existem demasiadas conclusões a retirar das consultas realizadas neste sistema. O *SQL Server* responde sem problemas às várias consultas realizadas. O desenho da consulta D tinha por objetivo colocar algumas dificuldades aos vários sistemas. Nesta consulta o *SQL Server* portou-se bastante bem, apesar de ter demorado cerca de um minuto e vinte segundos a responder. No segundo gráfico apresentado, descuram-se os resultados da consulta D, para se tentar focar as pequenas diferenças entre as outras três consultas. A ilustração permite aferir que não existe muito tempo a separar as consultas A, B e C. Todas possuem tempos de execução relativamente curtos, sendo que a evolução, à medida que se adicionam mais linhas, é bastante linear e previsível, portanto conclui-se que o *SQL Server* reage positivamente à adição de mais linhas ao sistema.

7.3 Resultados dos testes de performance: *MongoDB*

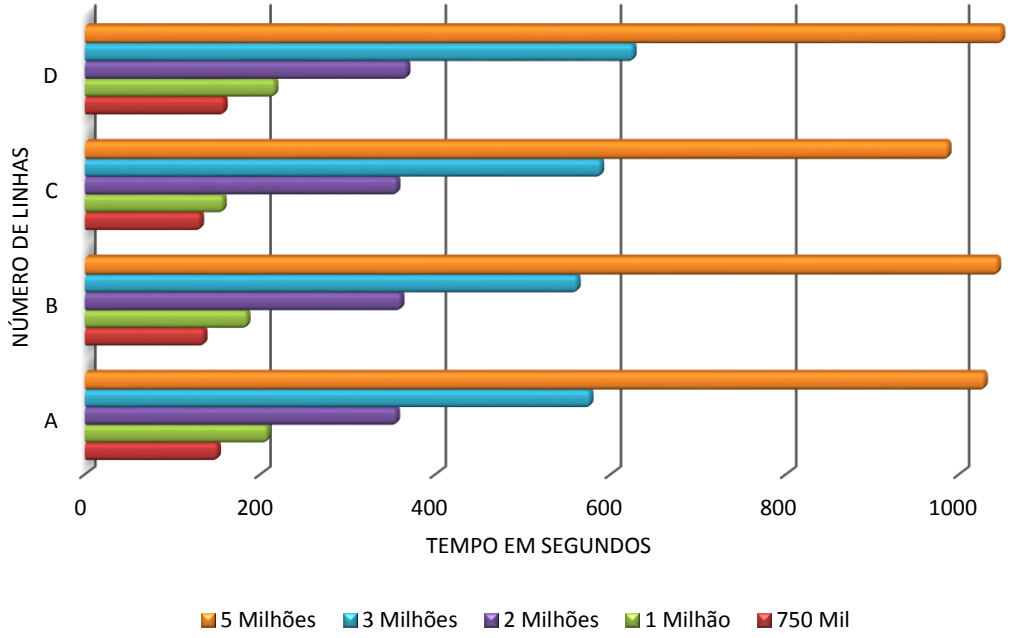

Figura 42 – Gráfico ilustrativo do tempo de resposta do sistema MongoDB a cada uma das consultas

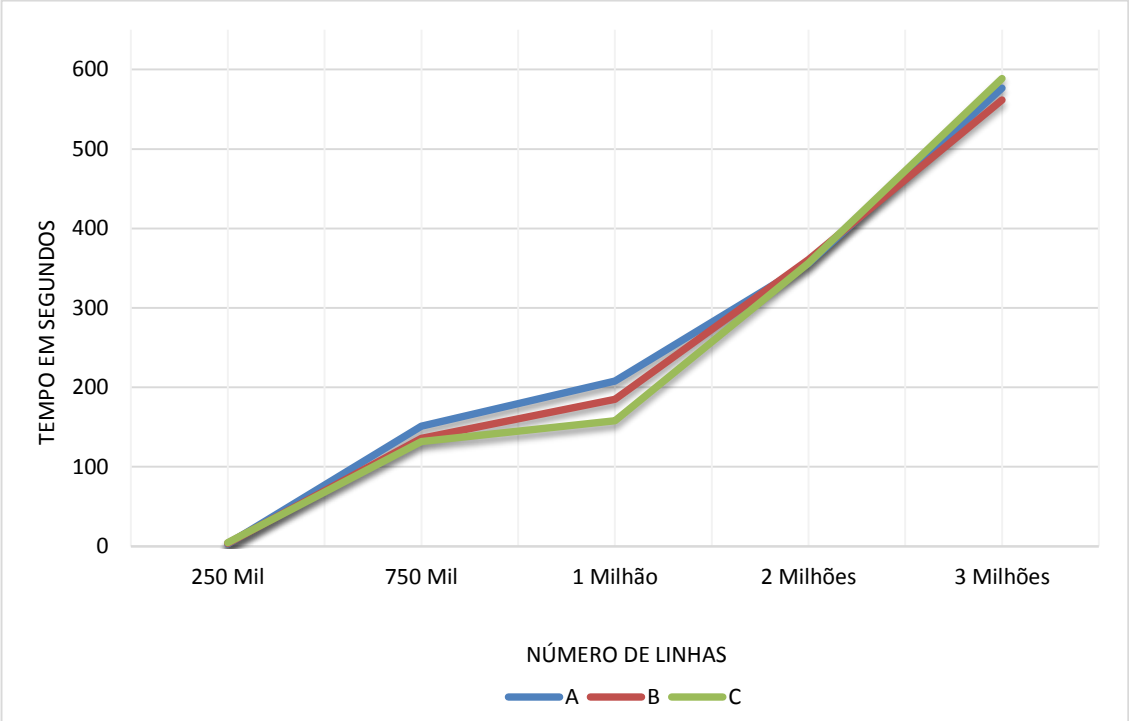

Figura 43 – Gráfico ilustrativo da evolução do tempo de resposta com o aumento de volume de dados

Tabela 12 - Tempo de resposta (em milissegundos) do sistema MongoDB a cada consulta

	Consulta A	Consulta B	Consulta C	Consulta D
250 Mil	2948	3700	4279	7074
750 Mil	151213	135871	131854	158859
1 Milhão	207959	184897	157762	216845
2 Milhões	355591	360663	355967	367532
3 Milhões	576538	561823	588697	625748
5 Milhões	1029688	1045168	987302	1052816

As consultas realizadas neste sistema tiveram por base o mesmo sistema operativo utilizado nas consultas *SQL Server*. Este testes foram realizados recorrendo à versão 2.6.4 do sistema MongoDB, cujo lançamento data de 11 de Agosto de 2014. Os testes foram realizados através da aplicação "*Mongo Shell*", porque este é o método mais rápido de interação com o sistema. Outro motivo que contribuiu para esta situação foi o facto de a serialização das consultas através do driver *C#* implicar uma alteração e reformulação do "código" das consultas, para que estas obedecessem aos parâmetros de serialização definidos. Informações sobre o "*Mongo Shell*" e outros interfaces do sistema *MongoDB* podem ser consultadas no Anexo A deste documento.

Os resultados das consultas realizadas com um volume de dados de 250 mil linhas não estão presentes no gráfico apresentado (Figura 42). Esta situação deve-se à grande diferença nos resultados obtidos em relação aos outros volumes. Os resultados no sistema MongoDB foram no geral bastante dececionantes. Apenas nas consultas com o menor volume de dados se obteve resultados (minimamente) aceitáveis. Uma das conclusões que se chega ao observar estes resultados é que o sistema MongoDB não é o mais indicado para responder a consultas que requeiram agregação de dados em tempo real. Independentemente do volume de dados, o esquema desenhado para este sistema teve um peso bastante elevado nos resultados obtidos. Gerir e calcular resultados de funções de agregação com vários documentos embutidos, cada um contendo vários atributos, acabou por se revelar um processo demasiado pesado para este sistema.

Apenas uma consulta realizada, sobre um volume de dados de 5 milhões, não ultrapassou o tempo de resposta de 15 minutos. Este é um valor demasiado elevado para a consulta em causa. Outra situação de referência, e que pode ser observada no segundo gráfico, é o facto de a consulta C obter o resultado mais rápido em diferentes volumes de dados. Esta situação não era esperada inicialmente, e em grande parte deve ser originada pela metodologia utilizada pelo sistema *MongoDB* para calcular os campos pedidos. Numa situação semelhante, a consulta A é, em diferentes volumes de dados, mais lenta que as consultas B e C, uma situação também não esperada, visto a consulta A ser a menos complexa de todas. Outra ilação tirada do primeiro gráfico apresentado é o simples facto de todas as consultas apresentam resultados bastante semelhantes para um volume de 2 milhões de linhas (Figura 42). Algo que não acontece em nenhum dos testes realizados no *SQL Server*, em que existe uma clara diferença de performance entre cada uma das diferentes consultas.

7.4 Resultados dos testes de performance: *Cassandra / Hive*

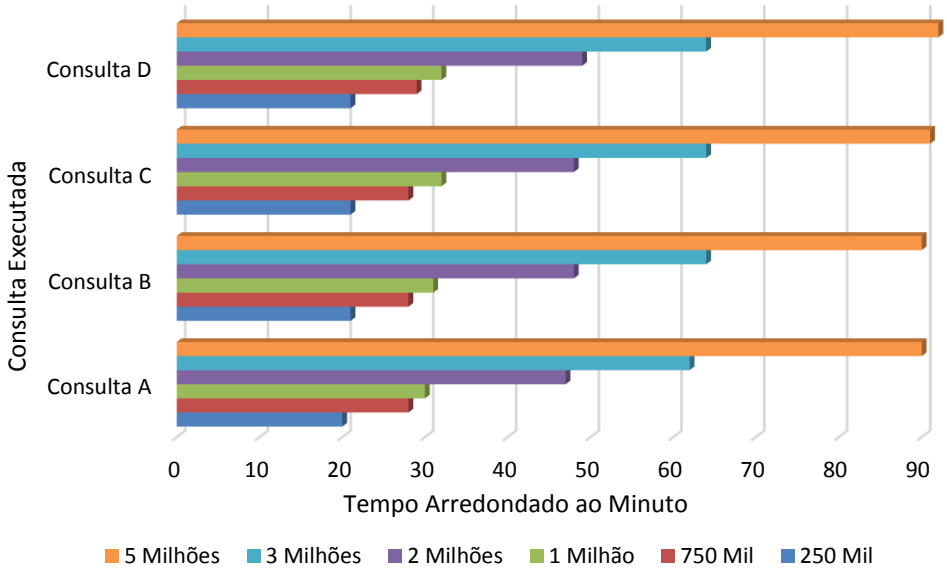

Figura 44 - Gráfico ilustrativo do tempo de resposta do sistema Cassandra a cada uma das consultas

Tabela 13 - Tempo de resposta (em segundos) do sistema *Cassandra/Hive* a cada consulta

	Consulta A	Consulta B	Consulta C	Consulta D
250 Mil	1196	1232	1234	1257
750 Mil	1665	1697	1708	1740
1 Milhão	1802	1830	1939	1944
2 Milhões	2840	2850	2909	2951
3 Milhões	3749	3836	3849	3856
5 Milhões	5378	5418	5446	5513

Os resultados obtidos com estes testes de performance são quase inacreditáveis. Não existem muitas conclusões a recolher de tais números. É, todavia, de salientar o facto de os resultados serem bastante consistentes ao longo das várias consultas. Todas as consultas apresentam um tempo de execução bastante semelhante em todos os volumes de dados.

Estes maus resultados devem-se, em grande parte, à natureza das consultas *Map Reduce*, que são o elemento principal da linguagem *HiveQL*. No momento da execução, o sistema *Hive* transforma as consultas de uma linguagem, virtualmente SQL, em complexas funções *Map Reduce*. No entanto, existe uma grande diferença entre a utilização tradicional do sistema *Hive* e aquela utilizada nestas consultas. Na sua utilização independente, o sistema *Hive* recorre ao armazenamento de dados no sistema de ficheiros da *framework Hadoop*, o *HDFS*. No caso aqui exposto, as consultas *Hive* estão a ser executadas sobre uma base de dados Cassandra, sendo que o sistema *Hive* e as consultas *HiveQL* não estão otimizados para operar sobre este tipo de armazenamento. Neste caso específico, a camada de abstração, criada pela DataStax e que se

posiciona entre o sistema *Hive* e a estrutura de armazenamento Cassandra, permite que sejam efetivamente executadas consultas *HiveQL* neste sistema, mas esta camada acaba por implicar um elevado custo de performance, gasto em todos os processos de conversão entre os dois sistemas. De salientar ainda que o facto de o sistema DataStax Enterprise estar a ser executado num sistema Linux, mas o seu armazenamento estar alocado num sistema de ficheiros *Windows* (NTFS), pode ter influenciado negativamente os resultados obtidos.

7.5 Conclusões: SQL Server VS MongoDB

Tabela 14 – Comparação entre o tempo de resposta (em segundos) dos dois sistemas

	A		B		C		D	
	SQL Server	MongoDB	SQL Server	MongoDB	SQL Server	MongoDB	SQL Server	MongoDB
250 Mil	0,9	2,9	1,2	3,7	1,5	4,3	3,2	7,1
750 Mil	2,6	151,2	3	135,9	3,3	131,9	7,6	158,9
1 Milhão	2,9	208	3,1	184,9	3,6	157,8	12,2	216,8
2 Milhões	5,9	355,6	6,1	360,7	6,6	356	32	367,5
3 Milhões	8,8	576,5	9	561,8	9,5	588,7	42,6	625,7
5 Milhões	13,9	1029,7	14,7	1045,2	16	987,3	80,5	1052,8

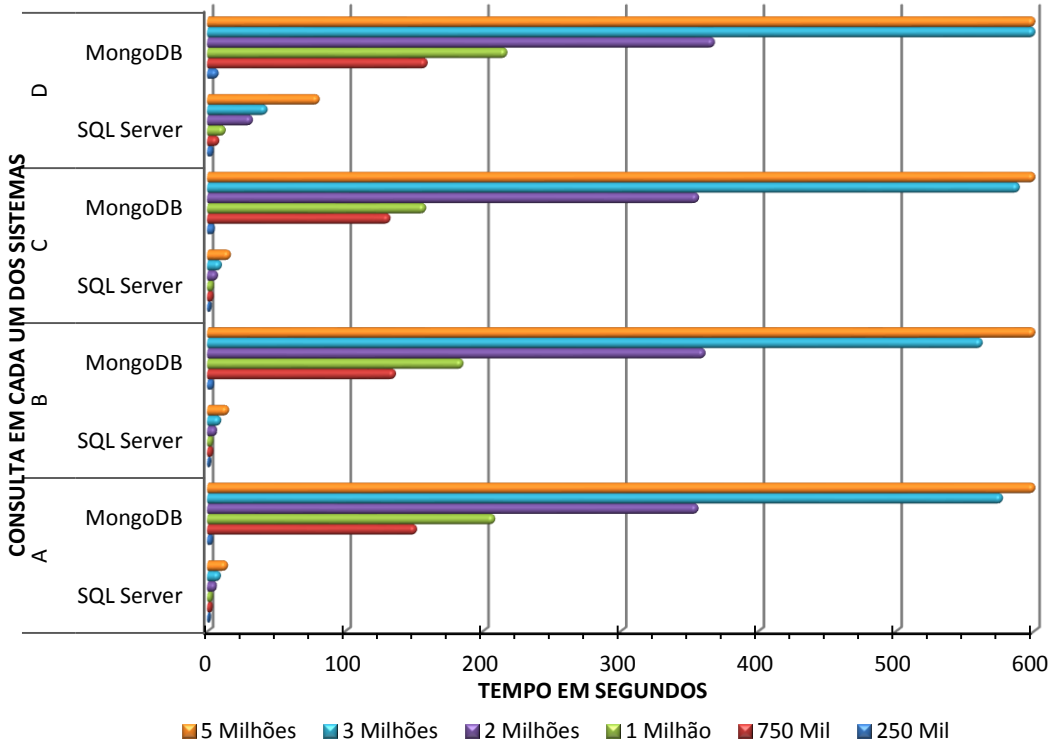

Figura 45 – Gráfico ilustrativo dos resultados obtidos para os sistemas SQL Server e MongoDB

Os resultados apresentados não permitem tirar outra conclusão que não seja o facto de o sistema *SQL Server* apresentar os melhores resultados, com uma elevada margem de vantagem. O volumes de dados utilizados não representam qualquer problema para o *SQL Server*, ficando bem longe dos resultados pré-históricos alcançados pelos outros sistemas. Os resultados obtidos no sistema *Cassandra* não entram no processo de comparação devido à sua elevada discrepância em relação aos outros sistemas.

As bases de dados *NoSQL* foram desenhadas com o intuito de escalar horizontalmente. Ao remover os fatores inerentes ao armazenamento distribuído está a retirar-se aos sistemas *NoSQL* o seu habitat natural e, por consequência, a desperdiçar a sua verdadeira utilidade. Nesta dissertação os sistemas foram simplesmente utilizados numa máquina local, sem distribuição de dados por outros nós. Apesar de parecer injusto colocar estes sistemas num ambiente que não o máximo das suas caraterísticas, os armazéns de dados são tradicionalmente hospedados numa única máquina, por consequência essa teria de ser a "configuração" a seguir.

É observável que nas consultas realizadas com o menor volume de dados, o sistema MongoDB apresentou resultados não muito distantes daqueles estabelecidos pelo *SQL Server*, o que permite concluir que o elevado número de documentos na mesma coleção acabou por gerar elevados problemas a este sistema. Sendo mais adequado utilizar várias coleções sem documentos embutidos do que apenas uma com vários documentos embutidos. Esta situação implica documentos maiores que, consequentemente, ocupam mais espaço contíguo em disco. Os esquemas de ambos os sistemas *NoSQL* utilizados tirariam muito mais vantagens se combinados com "junções" realizadas ao nível da lógica de uma aplicação. Não era, no entanto, adequado ao que se pretendia com esta dissertação recorrer a essa solução. Para representar a verdadeira essência das consultas num armazém de dados, as respostas teriam de ser obtidas através de uma única consulta, sem ser necessário escrever código ou criar aplicações. Crê-se veemente que os esquemas desenhados são os únicos capazes de responder na plenitude ao problema proposto.

É muito importante que os resultados obtidos sejam encarados apenas num contexto académico. Com outras regras e objetivos a alcançar, os resultados poderiam ser bastante distintos dos obtidos. As diferenças estratosféricas entre os vários resultados obtidos deve-se especialmente à incapacidade da máquina em causa, mais especificamente ao disco rígido utilizado. As consequências do que estes resultados significam para o enunciado estabelecido nesta dissertação serão abordadas no epílogo deste documento.

8 Conclusão

8.1 Síntese: Investigação concluída

A cada vez maior presença de dados *Big Data* na realidade empresarial atual levou a uma mudança nos algoritmos, paradigmas e tecnologias tradicionalmente utilizados no armazenamento e análise de dados. Neste projeto foi possível concluir que o conceito *Big Data* é bastante abrangente e que vários tipos de dados podem ser catalogados sob este conceito. Existem cada vez mais tecnologias preparadas para suprimir todas as necessidades tecnológicas geradas por dados gerados a uma alta velocidade, com origens nas mais variadas fontes e que se apresentam num formato não tradicional. Assim, nesta dissertação começou-se por entender quais as componentes que caraterizam estes dados. O estudo elaborado serviu de base a todos os restantes processos elaborados nesta dissertação. Este estudo, apesar de não aplicável num contexto prático, representa muito mais que cultura geral, representa conhecimento aplicável no agora em preparação para o futuro.

Ao mesmo tempo que os dados e tecnologias *Big Data* prosperam, os armazéns de dados tradicionais começam a apresentar dificuldades para suportar o imenso tamanho dos dados históricos das organizações. A realidade *Big Data* começa lentamente a alastrar aos armazéns de dados. Análises de dados que conjugam dados operacionais e dados *Big Data* começam a ser requeridas. O conhecimento e introspeção a tirar de uma tal abordagem são demasiado aliciantes para serem negligenciados pelas organizações. No estudo realizado no segundo capítulo deste documento é fornecida uma pequena introdução a este novo desafio para os armazéns de dados.

De entre as tecnologias *Big Data* salienta-se o movimento *NoSQL*. Este novo paradigma de armazenamento propõe-se a resolver os problemas inerentes ao armazenamento de dados *Big Data*. Um estudo profundo foi realizado de modo a alcançar o máximo de informação possível

sobre este tema. Apesar de ser um tema invariavelmente interessante, algumas restrições tiveram de ser colocadas à profundidade da investigação.

Na introdução desta dissertação, referiu-se a cada vez maior dificuldade apresentada pelas bases de dados relacionais para servirem de suporte a armazéns de dados. Esta investigação focava-se em responder à importante incerteza da viabilidade da utilização de bases de dados *NoSQL* em armazéns de dados. Para alcançar esse efeito executaram-se vários processos e estudos que culminaram na execução de várias consultas, tendo em vista medir a performance de cada uma. Essas consultas foram realizadas no sistema *MongoDB*, *Cassandra* e *SQL Server*. Todos estes sistemas apresentam diferentes tipos de armazenamento bem como diferentes metodologias de desenho e modelação de dados. Todos estes fatores acabaram por resultar numa investigação que colocou bastantes desafios mas, ao mesmo tempo, revelou-se bastante interessante.

Por fim nesta dissertação estudaram-se algumas ferramentas de *Business Intelligence* que possibilitam a integração e análise de dados *Big Data*. Este capítulo representa apenas um vislumbre das funcionalidades disponibilizadas por estas tecnologias. Este pequeno estudo realizado, deixou uma impressão bastante positiva, de modo a que é esperado que no futuro surjam novas ferramentas com funcionalidades melhores e aumentadas.

8.2 Discussão: Armazéns de dados em bases de dados *NoSQL*?

O primeiro ponto a salientar nesta discussão é a grande desilusão que representam os resultados obtidos. Esta desilusão só pode estar marcada a um nível psicológico, a investigação que se realizou foi considerada um sucesso. Não era objetivo desta dissertação obter resultados favoráveis para as bases de dados *NoSQL*. O objetivo era simplesmente obter resultados quantificáveis e comparáveis. O estudo que se realizou para se conseguir implementar um armazém de dados em cada sistema selecionado é bem mais importante para este projeto que os resultados obtidos. No entanto, esta investigação deparou-se com uma situação David versus Golias, em que as recentes e intrigantes bases de dados NoSQL defrontam o gigante *SQL Server*, instituído em 1989, no seu campo de batalha favorito.

Esta dissertação não tinha por objetivo obter como resultado uma resposta numa única palavra. Mas, à luz dos resultados obtidos, a resposta à questão "armazéns de dados em *NoSQL*?" terá sempre de ser só uma… Não! Vários fatores já abordados podem ter contribuído para os resultados obtidos, mas os tempos jurássicos destes testes não podem ser negligenciados.

No entanto, a resposta ao objetivo principal desta dissertação pode ser encarada de maneira diferente. É realmente viável utilizar bases de dados *NoSQL* em armazéns de dados? A resposta dada no parágrafo anterior indica que a resposta tende para um sólido não, mas essa visão está enclausurada num ambiente bastante restrito. As bases de dados *NoSQL* brilham no seu poder de escalar horizontalmente. É verdade que o volume de dados utilizado não causou nunca problemas ao *SQL Server*, mas a realidade é que na investigação realizada foi impossível atingir volumes de dados na ordem dos *terabytes*. Não é possível concluir portanto se as bases de

dados *NoSQL* com dados corretamente distribuídos por vários nós não representam uma verdadeira concorrência para as bases de dados relacionais neste contexto. Os preços praticados pelo *hardware* e software necessários para cada sistema não foram nunca parte desta investigação, mas é garantido que um sistema *NoSQL,* baseado em vários nós de hardware "comum", seria menos dispendioso que um super computador necessário para armazenar e gerir *terabytes* de dados num sistema relacional. Para algumas organizações, o custo extra a pagar poderá justificar a utilização de bases de dados *NoSQL* e o respetivo atraso na performance das análises. Do ponto de vista da viabilidade da utilização de NoSQL em armazéns de dados, pode-se concluir que é viável em todos os contextos, exceto no mais importante: as análises aos dados. Um fator que pode mudar bastante com a utilização de *hardware* atual, conjugado com as propriedades de escalabilidades das bases de dados *NoSQL*.

8.3 Considerações sobre os objetivos alcançados

Considera-se que esta investigação foi bem-sucedida tanto a um nível pessoal como académico. Os objetivos primários definidos foram todos alcançados com sucesso. No entanto, para se conseguir alcançar esses objetivos foi necessário flexibilizar alguns dos objetivos secundários definidos ao longo da investigação. Outro fator digno de ser salientado é o grande esforço e tempo gasto em processos secundários que visavam estudar um elevado número de questões não documentadas nesta dissertação.

Os objetivos alcançados contribuíram para o maior conhecimento do mundo *Big Data* em geral e das ferramentas *NoSQL* em particular. É esperado que grandes dividendos sejam obtidos, a nível pessoal, dos conhecimentos adquiridos ao longo desta investigação.

8.4 Trabalho futuro

É impossível deixar este projeto sem um grande grupo de novas questões e curiosidades por satisfazer. No decorrer desta dissertação entrou-se em contato com um grupo variado de novos conceitos em tecnologias, que infelizmente estavam presentes num plano lateral ao âmbito desta dissertação. Um esforço pessoal extra será empregue, para alcançar um melhor conhecimento sobre bases de dados *NoSQL*, *NewSQL* e tecnologias *Big Data* em geral.

Há, no entanto, um trabalho que necessita de ser continuado com esta dissertação. Diferentes comparações devem ser realizadas entre as bases de dados relacionais, *NoSQL* e *NewSQL*, de modo a aferir quais são as mais indicadas para vários casos de utilização. É uma sugestão interessante para futuros temas de dissertações e investigações académicas. No contexto específico da investigação desenvolvida, pensa-se que uma tarefa importante ficou por concretizar. Realizar testes de performance sobre um armazém de dados alocado num super computador, e armazéns de dados alocados em sistemas *MongoDB* e *Cassandra* escalados através de vários nós, sendo armazenados *terabytes* de dados em cada um. Fica alguma mágoa por ter sido impossível realizar estas operações na investigação atual.

Na investigação realizada sobre o tema OLAP em dados Big Data foi possível observar que é apenas uma questão de tempo até se tornar vital para uma organização conseguir integrar nos mesmos repositórios dados organizacionais e dados Big Data. Neste trabalho de mestrado não sobrou tempo, nem espaço suficiente para se estudar aprofundadamente este tema. No Anexo B, está presente um estudo realizado com o intuito de avaliar as capacidades de uma ferramenta desenhada para integrar e realizar análises OLAP sobre dados *Big Data*. Este estudo apesar de interessante, revelou-se bastante curto, pelo que se tomou a decisão de o remeter para anexo. Assim também se espera que no futuro seja possível realizar um estudo mais detalhado sobre este tema para, desse modo, completar os conhecimentos no sistema apresentado no referido anexo. Também seria extremamente interessante testar e comparar as funcionalidades de vários sistemas no que diz respeito ao âmbito ETL e OLAP sobre Big Data. Apesar de remetidos para um plano secundário, o tema e ferramenta estudados no Anexo B contribuem para um dos mais interessantes "projetos a realizar no futuro", resultantes de ambições e curiosidades não exploradas ao longo do desenvolvimento deste trabalho de mestrado.

Referências Bibliográficas

ABELL, A.; FERRARONS, J. & ROMERO, O. 978-1-4503-0963-9 Building cubes with MapReduce. Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP, 2011 Glasgow, Escócia, Reino Unidos. 2064680: ACM, 17-24, DOI: 10.1145/2064676.2064680.

AMAZON. 2014a. *Amazon DynamoDB* [Online]. Disponível em: https://aws.amazon.com/pt/dynamodb/ [Acedido em: 3 de Abril de 2014].

AMAZON. 2014b. *Amazon EMR* [Online]. Disponível em: http://aws.amazon.com/pt/elasticmapreduce/ [Acedido em: 26 de Julho de 2014].

ANGLES, R. & GUTIERREZ, C. 2008. Survey of graph database models. *ACM Comput. Surv.,* 40**,** 1-39, ISSN: 0360-0300, DOI: 10.1145/1322432.1322433.

ANIELLO, L.; BONOMI, S.; BRENO, M. & BALDONI, R. Assessing data availability of Cassandra in the presence of non-accurate membership. Proceedings of the 2nd International Workshop on Dependability Issues in Cloud Computing, 2013 Braga, Portugal. ACM, 1-6, DOI: 10.1145/2506155.2506157.

APACHE. 2014a. *Apache CoucheDB: Relax* [Online]. Disponível em: http://couchdb.apache.org/ [Acedido em: 3 de Abril de 2014].

APACHE. 2014b. *The Apache Hive™ data warehouse software* [Online]. Disponível em: https://hive.apache.org/ [Acedido em: 1 de Setembro de 2014].

APACHE. 2014c. *Cassandra Query Language (CQL) v3.1.7* [Online]. Disponível em: https://cassandra.apache.org/doc/cql3/CQL.html [Acedido em: 20 de Agosto de 2014].

APACHE. 2014d. *Welcome to Apache Cassandra* [Online]. Disponível em: http://cassandra.apache.org/ [Acedido em: 2 de Abril de 2014].

APACHE. 2014e. *Welcome to Apache HBase* [Online]. Disponível em: https://hbase.apache.org/ [Acedido em: 2 de Abril de 2014].

ASLETT, M. 2011. *The 451 Group White Paper: How will the database incumbents respond to NoSQL and NewSQL?* [Online]. Disponível em: http://cs.brown.edu/courses/cs227/archives/2012/papers/newsql/aslett-newsql.pdf [Acedido em: 4 de Abril 2011].

ASLETT, M. 2014. *NoSQL LinkedIn Skills Index – June 2014* [Online]. Disponível em: http://blogs.the451group.com/information_management/2014/07/01/nosql-linkedin-skills-index-june-2014/ [Acedido em: 18 de Junho de 2014].

AURADKAR, A.; BOTEV, C.; DAS, S.; DEMAAGD, D.; FEINBERG, A.; GANTI, P.; GAO, L.; GHOSH, B.; GOPALAKRISHNA, K.; HARRIS, B.; KOSHY, J.; KRAWEZ, K.; KREPS, J.; LU, S.; NAGARAJ, S.; NARKHEDE, N.; PACHEV, S.; PERISIC, I.; QIAO, L.; QUIGGLE, T.; RAO, J.; SCHULMAN, B.; SEBASTIAN, A.; SEELIGER, O.; SILBERSTEIN, A.; SHKOLNIK, B.; SOMAN, C.; SUMBALY, R.; SURLAKER, K.; TOPIWALA, S.; TRAN, C.; VARADARAJAN, B.; WESTERMAN, J.; WHITE, Z.; ZHANG, D. & ZHANG, J. Data Infrastructure at LinkedIn. Data Engineering (ICDE), 28th International Conference on, 1-5 Abril 2012 Washington DC, Estados Unidos da América. IEEE Computer Society, 1370-1381, ISSN: 1063-6382, DOI: 10.1109/ICDE.2012.147.

AURELIUS. 2014a. *The Benefits of Titan* [Online]. Disponível em: https://github.com/thinkaurelius/titan/wiki/The-Benefits-of-Titan [Acedido em: 3 de Abril de 2014].

AURELIUS. 2014b. *FAUNUS: Graph Analytics Engine* [Online]. Disponível em: http://thinkaurelius.github.io/faunus/ [Acedido em: 20 de Julho de 2014].

AURELIUS. 2014c. *TITAN: Distributed Graph Database* [Online]. Disponível em: http://thinkaurelius.github.io/titan/ [Acedido em: 3 de Abril de 2014].

BAGADE, P.; CHANDRA, A. & DHENDE, A. B. Designing performance monitoring tool for NoSQL Cassandra distributed database. Education and e-Learning Innovations (ICEELI), 2012 International Conference on, 1-3 Julho 2012 Sousse, Tunísia. IEEE Computer Society, 1-5, DOI: 10.1109/ICEELI.2012.6360579.

BAKSHI, K. Considerations for big data: Architecture and approach. Aerospace Conference, 3-10 Março 2012 Big Sky, Montana, Estados Unidos da América. IEEE Computer Society, 1-7, ISSN: 1095-323X, DOI: 10.1109/AERO.2012.6187357.

BARBIERATO, E.; GRIBAUDO, M. & IACONO, M. Modeling apache hive based applications in big data architectures. Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools, 2013 Torino, Itália. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 30-38, DOI: 10.4108/icst.valuetools.2013.254398.

BARMPIS, K. & KOLOVOS, D. S. Comparative analysis of data persistence technologies for large-scale models. Proceedings of the 2012 Extreme Modeling Workshop, 2012 Innsbruck, Áustria. 2467314: ACM, 33-38, DOI: 10.1145/2467307.2467314.

BASHO. 2014. *Riak is an open source, distributed database.* [Online]. Disponível em: http://basho.com/riak/ [Acedido em: 3 de Abril de 2014].

BEERNAERT, L.; GOMES, P.; MATOS, M.; VILA, R. & OLIVEIRA, R. 978-1-4503-2075-7. Evaluating Cassandra as a manager of large file sets. Proceedings of the 3rd International Workshop on Cloud Data and Platforms, 2013 Praga, República Checa. ACM, 25-30, DOI: 10.1145/2460756.2460761.

BENEFICO, S.; GJECI, E.; GOMARASCA, R. G.; LEVER, E.; LOMBARDO, S.; ARDAGNA, D. & NITTO, E. D. Evaluation of the CAP Properties on Amazon SimpleDB and Windows Azure Table Storage. 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 9 Setembro 2012 Timisoara, Roménia. DOI: 10.1109/SYNASC.2012.60.

BERNARDINO, J. & ABRAMOVA, V. NoSQL Databases: MongoDB vs Cassandra. C3S2E '13 - International C* Conference on Computer Science and Software Engineering Julho 10 - 12, 2013 Porto, Portugal. ACM, DOI: 10.1145/2494444.2494447.

BERNERS-LEE, T.; BIZER, C. & HEATH, T. 2009. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS),* 5**,** 1-22, ISSN: 1552-6283, DOI: 10.4018/jswis.2009081901.

BONNET, L.; LAURENT, A.; SALA, M.; LAURENT, B. & SICARD, N. Reduce, You Say: What NoSQL Can Do for Data Aggregation and BI in Large Repositories. Proceedings of the 22nd International Workshop on Database and Expert Systems Applications, 2011 Toulouse, França. IEEE Computer Society, 483-488, DOI: 10.1109/dexa.2011.71.

BORKAR, V. R.; CAREY, M. J. & LI, C. 2012. Big Data Platforms: What's Next? *XRDS.* Nova York, Estados Unidos da América: ACM.

BORLAND, B. 2014. *Pentaho Analytics for MongoDB: Combine Pentaho Analytics and MongoDB to create powerful analysis and reporting solutions: 1ª Edição.* Birmingham, Reino Unido, Packt Publishing. ISBN: 978-1-78216-835-5

BREWER, E. A. Towards robust distributed systems. Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, 2000 Portland, Oregon, Estados Unidos da América. ACM, 7, DOI: 10.1145/343477.343502.

BUCKLE, S. 2012. *IBM developerWorks: Introduction to VoltDB - Use an in-memory, high performance database with Java code* [Online]. Disponível em: http://www.ibm.com/developerworks/opensource/library/os-voltdb/os-voltdb-pdf.pdf [Acedido em: 6 de Abril de 2014].

CAPRIOLO, E.; WAMPLER, D. & RUTHERGLEN, J. 2012. *Programming Hive: 1ª Edição.* Sebastopol, Califórnia, Estados Unidos da América, O'Reilly Media. ISBN: 978-1-449-31933-5

CARLSON, J. L. 2013. *Redis in Action: 1ª Edição.* Shelter Island, Nova York, Estados Unidos da América, Manning Publications. ISBN: 9781935182054

CARNIEL, A. C.; SÁ, A. D. A.; BRISIGHELLO, V. H. P.; RIBEIRO, M. X.; BUENO, R. & CIFERRI, R. R. Query processing over data warehouse using relational databases and NoSQL. Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana, 1-5 Outubro 2012 Medellín, Colômbia. IEEE, 1-9, DOI: 10.1109/CLEI.2012.6427228.

CARSTOIU, D.; CERNIAN, A. & OLTEANU, A. Hadoop Hbase-0.20.2 performance evaluation. New Trends in Information Science and Service Science (NISS), 4th International Conference on, 11-13 Maio 2010 Gyeongju, Coreia do Sul. IEEE Computer Society, 84-87.

CATTELL, R. 2011. Scalable SQL and NoSQL data stores. *SIGMOD Rec.,* 39**,** 12-27, ISSN: 0163-5808, DOI: 10.1145/1978915.1978919.

CELKO, J. 2014. *Joe Celko's Complete Guide to NoSQL: 1ª Edição.* Estados Unidos da América, Morgan Kaufmann / Elsevier. ISBN: 978-0-12-407192-6

CHANG, F.; DEAN, J.; GHEMAWAT, S.; HSIEH, W. C.; WALLACH, D. A.; BURROWS, M.; CHANDRA, T.; FIKES, A. & GRUBER, R. E. Bigtable: a distributed storage system for structured data. Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7, 2006 Seattle, Estados Unidos da América. USENIX Association, 15-15.

CHAUDHURI, S. & DAYAL, U. 1997. An overview of data warehousing and OLAP technology. *SIGMOD Rec.,* 26**,** 65-74, ISSN: 0163-5808, DOI: 10.1145/248603.248616.

CHEN, M.; MAO, S. & LIU, Y. 2014. Big Data: A Survey. *Mobile Networks and Applications,* 19**,** 171-209, ISSN: 1383-469X, DOI: 10.1007/s11036-013-0489-0.

CHEN, Z. & ORDONEZ, C. Efficient OLAP with UDFs. Proceedings of the ACM 11th international workshop on Data warehousing and OLAP, 2008 Napa Valley, Califórnia, Estados Unidos da América. ACM, 41-48, DOI: 10.1145/1458432.1458440.

CHODOROW, K. 2013. *MongoDB: The Definitive Guide: 2ª Edição.* Sebastopol, Califórnia, Estados Unidos da América O'Reilly Media, Inc. ISBN: 978-1-449-34468-9

CLARK, J. 2013. *Google goes back to the future with SQL F1 database* [Online]. Disponível em: http://www.theregister.co.uk/2013/08/30/google_f1_deepdive/ [Acedido em: 3 de MArço de 2014].

CLUSTRIX. 2012. *White Paper - A New Approach: Clustrix Sierra Database Engine* [Online]. Disponível em: http://www.clustrix.com/wp-content/uploads/2013/10/Clustrix_A-New-Approach_WhitePaper.pdf [Acedido em: 7 de Abril de 2014].

CLUSTRIX. 2014. *ClustrixDB: Real Time. Real Scale. Any Cloud.* [Online]. Disponível em: http://www.clustrix.com/ [Acedido em: 7 de Abril de 2014].

CODD, E. F.; CODD, S. B. & SALLEY, C. T. 1993. *Providing OLAP to User-Analysts: An IT Mandate* [Online]. E.F. Codd Associates. Disponível em: http://www.minet.uni-jena.de/dbis/lehre/ss2005/sem_dwh/lit/Cod93.pdf [Acedido em: 9 de março de 2014].

COHEN, J.; DOLAN, B.; DUNLAP, M.; HELLERSTEIN, J. M. & WELTON, C. 2009. MAD skills: new analysis practices for big data. *Proc. VLDB Endow.,* 2**,** 1481-1492, ISSN: 2150-8097.

COPELAND, R. 2013. *MongoDB Applied Design Patterns: 1ª Edição.* Sebastopol, Califórnia, Estados Unidos da América, O'Reilly Media, Inc. ISBN: 978-1-449-34004-9

CORBETT, J. C.; DEAN, J.; EPSTEIN, M.; FIKES, A.; FROST, C.; FURMAN, J.; GHEMAWAT, S.; GUBAREV, A.; HEISER, C.; HOCHSCHILD, P.; HSIEH, W.; KANTHAK, S.; EUGENE KOGAN, H. L.; LLOYD, A.; MELNIK, S.; MWAURA, D.; NAGLE, D.; QUINLAN, S.; RAO, R.; ROLIG, L.; SAITO, Y.; SZYMANIAK, M.; TAYLOR, C. & RUTH WANG, D. W. Spanner: Google's Globally-Distributed Database. Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation OSDI'12, 2012 Hollywood, California, Estados Unidos da América. USENIX Association.

CUZZOCREA, A.; BELLATRECHE, L. & SONG, I.-Y. 978-1-4503-2412-0 Data warehousing and OLAP over big data: current challenges and future research directions. Proceedings of the sixteenth international workshop on Data warehousing and OLAP, 2013 San Francisco, Califólrnia, Estados Unidos da América. ACM, 67-70, DOI: 10.1145/2513190.2517828.

CUZZOCREA, A.; SONG, I.-Y. & DAVIS, K. C. 978-1-4503-0963-9. Analytics over large-scale multidimensional data: the big data revolution! Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP, 2011 Glasgow, Escócia, Reino Unido. ACM, 101-104, DOI: 10.1145/2064676.2064695.

DATASTAX. 2012. *Apache Cassandra 1.1 Documentation: Data Modeling* [Online]. Disponível em: http://www.datastax.com/docs/1.1/ddl/column_family [Acedido em: 20 de Agosto de 2014].

DATASTAX. 2014a. *About DataStax* [Online]. Disponível em: http://www.datastax.com/company [Acedido em: 15 de Agosto de 2014].

DATASTAX. 2014b. *C# Driver 2.1 for Apache Cassandra* [Online]. Disponível em: http://www.datastax.com/documentation/developer/csharp-driver/2.1/csharp-driver/whatsNew2.html [Acedido em: 21 de Agosto de 2014].

DATASTAX. 2014c. *CQL for Cassandra 2.x* [Online]. Disponível em: http://www.datastax.com/documentation/cql/3.1/cql/cql_intro_c.html [Acedido em: 22 de Agosto de 2014].

DATASTAX. 2014d. *DataStax Enterprise* [Online]. Disponível em: http://www.datastax.com/what-we-offer/products-services/datastax-enterprise [Acedido em: 1 de Setembro de 2014].

DATASTAX. 2014e. *Planet Cassandra: A DataStax Community Service* [Online]. Disponível em: http://planetcassandra.org/cassandra/ [Acedido em: 24 de Agosto de 2014].

DBMS2. 2010. *Toward a NoSQL taxonomy* [Online]. Disponível em: http://www.dbms2.com/2010/03/14/nosql-taxonomy/ [Acedido em: 26 de Março de 2014 2014].

DBMS2. 2013. *The Clustrix story* [Online]. Disponível em: http://www.dbms2.com/2010/05/12/the-clustrix-story/ [Acedido em: 7 de Abril de 2014].

DEAN, J. & GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, Dezembro de 2004 San Francisco, Estados Unidos da América.

DECANDIA, G.; HASTORUN, D.; JAMPANI, M.; KAKULAPATI, G.; LAKSHMAN, A.; PILCHIN, A.; SIVASUBRAMANIAN, S.; VOSSHALL, P. & VOGELS, W. 2007. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.,* 41**,** 205-220, ISSN: 0163-5980, DOI: 10.1145/1323293.1294281.

DEDE, E.; GOVINDARAJU, M.; GUNTER, D.; CANON, R. S. & RAMAKRISHNAN, L. Performance evaluation of a MongoDB and hadoop platform for scientific data analysis. Proceedings of the 4th ACM workshop on Scientific cloud computing, 2013 Nova York, Estados Unidos da América. ACM, 13-20, DOI: 10.1145/2465848.2465849.

DEMCHENKO, Y.; GROSSO, P.; LAAT, C. D. & MEMBREY, P. Addressing Big Data Issues in Scientific Data Infrastructure. Collaboration Technologies and Systems (CTS), 2013 International Conference on, 20-24 Maio 2013 San Diego, California, Estados Unidos da América. IEEE Computer Society, 48 - 55, DOI: 10.1109/CTS.2013.6567203.

DHARMASIRI, H. M. L. & GOONETILLAKE, M. D. J. S. A federated approach on heterogeneous NoSQL data stores. Advances in ICT for Emerging Regions (ICTer), 2013 International Conference on, 11-15 Dezembro 2013 Colombo, Sri lanka. IEEE, 234-239, DOI: 10.1109/ICTer.2013.6761184.

DIMIDUK, N. & KHURANA, A. 2013. *HBase in Action: 1ª Edição.* Shelter Island, Nova York, Estados Unidos da América, Manning Publications. ISBN: 9781617290527

DOSHI, K. A.; ZHONG, T.; LU, Z.; TANG, X.; LOU, T. & DENG, G. Blending SQL and NewSQL Approaches: Reference Architectures for Enterprise Big Data Challenges. Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on, 10-12 Outubro 2013 Pequim, china. IEEE Computer Society, 163-170, DOI: 10.1109/CyberC.2013.34.

EDLICH, S. 2014. *Your Ultimate Guide to the Non-Relational Universe!* [Online]. Disponível em: http://nosql-database.org/ [Acedido em: 25 de Março de 2014].

FINK, B. 978-1-4503-1575-3. Distributed computation on dynamo-style distributed storage: riak pipe. Proceedings of the eleventh ACM SIGPLAN workshop on Erlang workshop, 2012 Copenhaga, Dinamarca. ACM, 43-50, DOI: 10.1145/2364489.2364497.

GAO, X.; NACHANKAR, V. & QIU, J. 978-1-4503-1157-1. Experimenting lucene index on HBase in an HPC environment. Proceedings of the first annual workshop on High performance computing meets databases, 2011 Seattle, Washington, Estados Unidos da América. ACM, 25-28, DOI: 10.1145/2125636.2125646.

GARULLI, L. 2014. *OrientDB Announcement: Updated Distributed Architecture and new Sharding feature* [Online]. Disponível em: http://www.orientechnologies.com/distributed-architecture-sharding/ [Acedido em: 29 de Julho de 2014].

GEORGE, L. 2011. *HBase: The Definitive Guide: 1ª Edição.* Sebastopol, Califórnia, Estados Unidos da América, O'Reilly Media, Inc. ISBN: 978-1-449-39610-7

GERHARDT, B.; GRIFFIN, K. & KLEMANN, R. 2012. *CISCO: Unlocking Value in the Fragmented World of Big Data Analytics: How Information Infomediaries Will Create a New Data Ecosystem* [Online]. Disponível em: https://www.cisco.com/web/about/ac79/docs/sp/Information-Infomediaries.pdf [Acedido em: 5 de Setembro de 2014].

GHEMAWAT, S.; GOBIOFF, H. & LEUNG, S.-T. The Google file system. 19th ACM Symposium on Operating Systems Principles, 2003 Lake George, Nova York, Estados Unidos da América. ACM, DOI: 10.1145/945445.945450.

GILBERT, S. & LYNCH, N. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News,* Vol. 33**,** 9, ISSN: 0163-5700, DOI: 10.1145/564585.564601.

GILBERT, S. & LYNCH, N. A. 2012. Perspectives on the CAP Theorem. *Computer,* vol. 45 6, DOI: 10.1109/MC.2011.389.

GROLINGER, K.; HIGASHINO, W. A.; TIWARI, A. & CAPRETZ, M. A. 2013. Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications,* 2, ISSN: 2192-113X, DOI: 10.1186/2192-113X-2-22.

GROUP, T. 2014. *451 Research* [Online]. Disponível em: https://451research.com/ [Acedido em: 4 de Março de 2014].

HAINES, K. 2009. *Engine Yard Blog: To Redis or Not To Redis?* [Online]. Disponível em: https://blog.engineyard.com/2009/key-value-stores-for-ruby-part-4-to-redis-or-not-to-redis [Acedido em: 3 de Abril de 2014].

HAN, J.; E, H.; LE, G. & DU, J. Survey on NoSQL database. 6th International Conference on Pervasive Computing and Applications, 26-28 Outubro 2011 2011 Port Elizabeth, África do Sul. IEEE Computer Society, 3, ISSN, DOI: 10.1109/ICPCA.2011.6106531.

HECHT, R. & JABLONSKI, S. NoSQL evaluation: A use case oriented survey. Proceedings of the 2011 International Conference on Cloud and Service Computing, 2011 Hong Kong, China. IEEE Computer Society, 336-341, DOI: 10.1109/csc.2011.6138544.

HIBERNATINGRHINOS. 2014. *RavenDB: Second Generation Document DB* [Online]. Disponível em: http://ravendb.net/ [Acedido em: 3 de Abril de 2014].

HIGGINBOTHAM, S. 2010. *Clustrix Builds the Webscale Holy Grail: A Database That Scales* [Online]. Disponível em: http://gigaom.com/2010/05/03/clustrix-builds-the-webscale-holy-grail-a-database-that-scales/ [Acedido em: 7 de Abril de 2014].

HOLMES, A. 2012. *Hadoop in Practice: 1ª Edição.* Estados Unidos da América, Manning Publications. ISBN: 9781617290237

HUAI, Y.; CHAUHAN, A.; GATES, A.; HAGLEITNER, G.; HANSON, E. N.; O'MALLEY, O.; PANDEY, J.; YUAN, Y.; LEE, R. & ZHANG, X. Major technical advancements in apache hive. Proceedings of the 2014 ACM SIGMOD international conference on Management of data, 2014 Snowbird, Utah, Estados Unidos da América. ACM, 1235-1246, DOI: 10.1145/2588555.2595630.

HURWITZ, J.; NUGENT, A.; HALPER, F. & KAUFMAN, M. 2013. *Big Data For Dummies: 1ª Edição.* Hoboken, New Jersey, Estados Unidos da América, John Wiley & Sons, Inc. ISBN: 978-1-118-50422-2

HYPERTABLE. 2014. *HYPERTABLE INC: Big Data, Big Performance* [Online]. Disponível em: http://hypertable.org/ [Acedido em: 2 de Abril de 2014].

IMHOFF, C.; GALEMMO, N. & GEIGER, J. G. 2003. *Mastering Data Warehouse Design: Relational and Dimensional Techniques: 1ª Edição.* Indianapolis, Indiana, Estados Unidos da América, Wiley Publishing, Inc. ISBN: 0-471-32421-3

INDRAWAN-SANTIAGO, M. Database Research: Are We at a Crossroad? Reflection on NoSQL. Proceedings of the 2012 15th International Conference on Network-Based Information Systems, 2012 Melbourne, Austrália. IEEE Computer Society, 45-51, DOI: 10.1109/NBiS.2012.95.

INMON, W. H. 1992. *Building the Data Warehouse: 1ª Edição.* Estados Unidos da América, John Wiley & Sons, Inc. ISBN: 0471569607

JORGENSEN, A.; ROWLAND-JONES, J.; WELCH, J.; CLARK, D.; PRICE, C. & MITCHELL, B. 2014. *Microsoft® Big Data Solutions: 1ª Edição.* Indianápolis, Indiana, Estados Unidos da América, John Wiley & Sons,. ISBN: 978-1-118-72908-3

KALLMAN, R.; KIMURA, H.; NATKINS, J.; PAVLO, A.; RASIN, A.; ZDONIK, S.; JONES, E. P. C.; MADDEN, S.; STONEBRAKER, M.; ZHANG, Y.; HUGG, J. & ABADI, D. J. 2008. H-store: a high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.,* 1**,** 1496-1499, ISSN: 2150-8097.

KAUR, K. & RANI, R. Modeling and querying data in NoSQL databases. Big Data, 2013 IEEE International Conference on, 6-9 Oct. 2013 2013 Silicon Valley, California, Estados Unidos da América. IEEE Computer Society, 1-7, ISSN, DOI: 10.1109/BigData.2013.6691765.

KEHAYIAS, J. 2012. *SQLskills: Enlarging the AdventureWorks Sample Databases* [Online]. Disponível em: http://www.sqlskills.com/blogs/jonathan/enlarging-the-adventureworks-sample-databases/ [Acedido em: 12 de Julho de 2014].

KIMBALL, R. & ROSS, M. 2013. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling: 3ª Edição.* Indianapolis, Indiana, Estados Unidos da América, John Wiley & Sons, Inc. ISBN: 978-1-118-53080-1

KOLOMICENKO, V.; SVOBODA, M. & HOLUBOVÁ, I. 978-1-4503-2113-6. Experimental Comparison of Graph Databases. Proceedings of International Conference on Information Integration and Web-based Applications & Services, 2013 Viena, Áustria. ACM, 115-124, DOI: 10.1145/2539150.2539155.

KONISHETTY, V. K.; KUMAR, K. A.; VORUGANTI, K. & RAO, G. V. P. 978-1-4503-1196-0. Implementation and evaluation of scalable data structure over HBase. Proceedings of the International Conference on Advances in Computing, Communications and Informatics, 2012 Chennai, Índia. ACM, 1010-1018, DOI: 10.1145/2345396.2345559.

KORLA, N. 2013. *SlideShare: Cassandra Data Modeling - Practical Considerations @ Netflix* [Online]. Disponível em: http://pt.slideshare.net/nkorla1share/cass-summit-3 [Acedido em: 21 de Agosto de 2014].

KRISHNAN, K. 2013. *Data Warehousing in the Age of Big Data: 1ª Edição.* Massachusetts, Estados Unidos da América, Elsevier. ISBN: 978-0-12-405891-0

LAKSHMAN, A. & MALIK, P. 2010. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.,* 44**,** 35-40, ISSN: 0163-5980, DOI: 10.1145/1773912.1773922.

LAM, C. 2011. *Hadoop in Action: 1ª Edição.* Estados Unidos da América, Manning Publications. ISBN: 9781935182191

LAPA, J.; BERNARDINO, J. & FIGUEIREDO, A. A comparative analysis of open source business intelligence platforms. Proceedings of the International Conference on Information Systems and Design of Communication, 2014 Lisboa, Portugal. ACM, 86-92, DOI: 10.1145/2618168.2618182.

LEBRESNE, S. 2012. *DataStax: A thrift to CQL3 upgrade guide* [Online]. Disponível em: http://www.datastax.com/dev/blog/thrift-to-cql3 [Acedido em: 20 de Agosto e 2014].

LEE, K. K.-Y.; TANG, W.-C. & CHOI, K.-S. 2013. Alternatives to relational database: Comparison of NoSQL and XML approaches for clinical data storage. *Comput. Methods Prog. Biomed.,* 110**,** 99-109, ISSN: 0169-2607, DOI: 10.1016/j.cmpb.2012.10.018.

LEONARD, A. 2013. *Pro Hibernate and MongoDB: 1ª Edição.* Nova York, Estados Unidos da América, Appress. ISBN: 978-1-4302-5794-3

LERNER, R. M. 2010a. At the forge: Cassandra. *Linux J.,* 2010**,** 7, ISSN: 1075-3583.

LERNER, R. M. 2010b. At the forge: Redis. *Linux J.,* 2010**,** 5, ISSN: 1075-3583.

LIM, E.-P.; CHEN, H. & CHEN, G. 2013. Business Intelligence and Analytics: Research Directions. *ACM Trans. Manage. Inf. Syst.,* 3**,** 1-10, ISSN: 2158-656X, DOI: 10.1145/2407740.2407741.

LIU, Y.; WANG, Y. & JIN, Y. Research on the improvement of MongoDB Auto-Sharding in cloud environment. Computer Science & Education (ICCSE), 2012 7th International Conference on, 14-17 July 2012 2012 Melbourne, austrália. IEEE Computer Society, 851-854, DOI: 10.1109/ICCSE.2012.6295203.

LOSHIN, D. 2013. *Big Data Analytics: From Strategic Planning to Enterprise Integration with Tools, Techniques, NoSQL, and Graph: 1ª Edição.* Estados Unidos da América, Morgan Kaufmann / Elsevier. ISBN: 978-0-12-417319-4

MACHANIC, A. 2011. *The SQL Server Blog Spot on the Web: Thinking Big (Adventure)* [Online]. Disponível em: http://sqlblog.com/blogs/adam_machanic/archive/2011/10/17/thinking-big-adventure.aspx [Acedido em: 12 de Julho de 2014].

MANSMANN, S.; REHMAN, N. U.; WEILER, A. & SCHOLL, M. H. 2014. Discovering OLAP dimensions in semi-structured data. *Information Systems,* 44**,** 120–133, ISSN: 0306-4379, DOI: 10.1016/j.is.2013.09.002.

MAO, Y.; KOHLER, E. & MORRIS, R. T. Cache craftiness for fast multicore key-value storage. Proceedings of the 7th ACM european conference on Computer Systems, 2012 Berna, Suíça. ACM, 183-196, DOI: 10.1145/2168836.2168855.

MCCREARY, D. & KELLY, A. 2014. *Making Sense of NoSQL: A Guide for Managers and the Rest of Us: 1ª Edição.* Shelter Island, Nova York, Estados Unidos da América, Manning Publications. ISBN: 9781617291074

MCMURTRY, D.; OAKLEY, A.; SHARP, J.; SUBRAMANIAN, M. & ZHANG, H. 2013. *Data Access for Highly-Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence: 1ª Edição.* Formato Digital por Microsft, Estados Unidos da América. ISBN: 978-1-62114-030-6

MICROSOFT. 2014a. *Adventure Works for SQL Server 2012* [Online]. Disponível em: http://msftdbprodsamples.codeplex.com/releases/view/55330 [Acedido em: 27 de Julho de 2014].

MICROSOFT. 2014b. *Get started with the HDInsight Emulator* [Online]. Disponível em: http://azure.microsoft.com/en-us/documentation/articles/hdinsight-get-started-emulator/ [Acedido em: 1 de Setembro de 2014].

MICROSOFT. 2014c. *HDInsight: O nosso serviço 100% baseado em Apache Hadoop na nuvem* [Online]. Disponível em: http://azure.microsoft.com/pt-pt/services/hdinsight/ [Acedido em: 1 de Setembro de 2014].

MICROSOFT. 2014d. *Microsoft Azure* [Online]. Disponível em: https://azure.microsoft.com/pt-pt/ [Acedido em: 1 de Setembro de 2014].

MICROSOFT. 2014e. *SQL Server: The foundation of Microsoft's comprehensive data platform* [Online]. Disponível em: http://www.microsoft.com/en-us/server-cloud/products/sql-server/default.aspx [Acedido em: 22 de Julho de 2014].

MINHAS, U. F.; LIU, R.; ABOULNAGA, A.; SALEM, K.; NG, J. & ROBERTSON, S. Elastic Scale-Out for Partition-Based Database Systems. Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops, 2012 Washington DC, Estados Unidos da América. IEEE Computer Society, 281-288, DOI: 10.1109/icdew.2012.52.

MIT-CSAIL. 2008. *People: Michael Stonebraker* [Online]. Disponível em: http://www.csail.mit.edu/user/1547].

MOHANTY, S.; JAGADEESH, M. & SRIVATSA, H. 2013. *Big Data Imperatives: Enterprise Big Data Warehouse, BI Implementations and Analytics: 1ª Edição.* Nova York, Estado Unidos da América, Apress. ISBN: 978-1-4302-4872-9

MONGODB. 2014a. *BSON Types* [Online]. Disponível em: http://docs.mongodb.org/manual/reference/bson-types/ [Acedido em: 17 de Agosto de 2014].

MONGODB. 2014b. *C# and .NET MongoDB Driver* [Online]. Disponível em: http://docs.mongodb.org/ecosystem/drivers/csharp/ [Acedido em: 17 de Agosto de 2014].

MONGODB. 2014c. *Mapping SQL to MongoDB* [Online]. Disponível em: http://info.mongodb.com/rs/mongodb/images/sql_to_mongo.pdf [Acedido em: 7 de Setembro de 2014].

MONGODB. 2014d. *The MongoDB 2.6 Manual* [Online]. Disponível em: http://docs.mongodb.org/manual/ [Acedido em: 8 de Setembro de 2014].

MONGODB. 2014e. *MongoDB CRUD Introduction* [Online]. Disponível em: http://docs.mongodb.org/manual/core/crud-introduction/ [Acedido em: 5 de Agosto de 2014].

MONGODB. 2014f. *MongoDB Documentation Project: Data Model Design for MongoDB - Release 2.6.4* [Online]. Disponível em: http://docs.mongodb.org/master/MongoDB-data-models-guide.pdf [Acedido em: 9 de Setembro de 2014].

MONGODB. 2014g. *MongoDB Downloads* [Online]. Disponível em: http://www.mongodb.org/downloads [Acedido em: 30 de Junho de 2014].

MONGODB. 2014h. *MongoDB: Aggregation Introduction* [Online]. Disponível em: http://docs.mongodb.org/manual/core/aggregation-introduction/ [Acedido em: 8 de Setembro de 2014].

MONGODB. 2014i. *MongoDB: Agile and Scalable* [Online]. Disponível em: http://www.mongodb.org/ [Acedido em: 2 de Abril de 2014].

MONGODB. 2014j. *RDBMS to MongoDB Migration Guide: Considerations and Best Practices (A MongoDB White Paper)* [Online]. Disponível em: http://info.mongodb.com/rs/mongodb/images/RDBMStoMongoDBMigration.pdf [Acedido em: 27 de Julho de 2014].

MONGOVUE. 2014. *MongoVUE: GUI Tools for MongoDB* [Online]. Disponível em: http://www.mongovue.com/ [Acedido em: 18 de Agosto de 2014].

MONIRUZZAMAN, A. B. M. & HOSSAIN, S. A. 2013. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *International Journal of Database Theory and Application,* Vol. 6**,** 14.

NAWROTH, A. 2010. *The top 10 ways to get to know Neo4j* [Online]. Disponível em: http://neo4j.com/blog/the-top-10-ways-to-get-to-know-neo4j/ [Acedido em: 3 de Abril de 2014].

NAYAK, A.; PORIYA, A. & POOJARY, D. 2013. Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems,* 5**,** 16-19, ISSN: 2249-0868, DOI: 10.5120/ijais12-450888.

NEERAJ, N. 2013. *Mastering Apache Cassandra: 1ª Edição.* Birmingham, Reino Unido, Packt Publishing. ISBN: 978-1-78216-268-1

NEO-TECHNOLOGY. 2014a. *Neo4J Graph Database* [Online]. Disponível em: http://www.neo4j.org/ [Acedido em: 3 de Abril de 2014].

NEO-TECHNOLOGY. 2014b. *What is Neo4j?* [Online]. Disponível em: http://www.neo4j.org/learn/neo4j [Acedido em: 3 de Abril de 2014].

NICOL, G. 2013. *Getting started with Hadoop on Cassandra* [Online]. Disponível em: http://gerrymcnicol.azurewebsites.net/ [Acedido em: 1 de Setembro de 2014].

NUODB. 2014. *NuoDB: The Distributed Database* [Online]. Disponível em: http://www.nuodb.com/explore/newsql-cloud-database-product [Acedido em: 7 de Abril de 2014].

O'NEIL, P.; O'NEIL, B. & CHEN, X. 2009. *Star Schema Benchmark: Revision 3, June 5, 2009* [Online]. Disponível em: http://www.cs.umb.edu/~poneil/StarSchemaB.pdf [Acedido em: 17 de Julho de 2014].

O'REILLY MEDIA, I. 2012. *Big Data Now: 2012 Edition: 1ª Edição.* Sebastopol, California, Estados Unidos da América, O'Reilly Media, Inc. ISBN: 978-1-449-35671-2

ORACLE. 2014. *Why Oracle Database 12c?* [Online]. Disponível em: https://www.oracle.com/database/index.html [Acedido em: 22 de Julho de 2014].

ORIENTECHNOLOGIES. 2014. *What is OrientDB?* [Online]. Disponível em: http://www.orientechnologies.com/orientdb/ [Acedido em: 3 de Abril de 2014].

PARALECT. 2014. *Robomongo: Shell-centric cross-platform MongoDB management tool* [Online]. Disponível em: http://robomongo.org/ [Acedido em: 17 de Agosto de 2014].

PARTHASARATHY, V. 2013. *Learning Cassandra for Administrators: Optimize high-scale data by tuning and troubleshooting using Cassandra: 1ª Edição.* Birmingham, Reimo Unido, Packt Publishing Ltd. ISBN: 978-1-78216-817-1

PATEL, A. B.; BIRLA, M. & NAIR, U. Addressing Big Data Problem Using Hadoop and Map Reduce. NUiCONE-Nirma University International Conference on Engineeroing, 06-08 Dezembro 2012, Ahmedabad, Gujarat, Índia. IEEE Computer Society, 1 - 5, DOI: 10.1109/NUICONE.2012.6493198.

PENTAHO. 2014a. *BLUEPRINT FOR BIG DATA SUCCESS: Monetize My Data* [Online]. Disponível em: http://www.pentaho.com/Monetize-My-Data [Acedido em: 3 de Outrubro de 2014].

PENTAHO. 2014b. *BLUEPRINT FOR BIG DATA SUCCESS: Streamlined Data Refinery* [Online]. Disponível em: http://www.pentaho.com/Streamlined-Data-Refinery [Acedido em: 3 de Outubro de 2014].

PENTAHO. 2014c. *Pentaho Big Data Analytics* [Online]. Disponível em: http://www.pentaho.com/product/big-data-analytics [Acedido em: 3 de Outubro de 2014].

PIRZADEH, P.; TATEMURA, J. & HACIGUMUS, H. Performance Evaluation of Range Queries in Key Value Stores. Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, 16-20 Maio 2011 Shanghai, China. IEEE Computer Society, 1092-1101, ISSN: 1530-2075, DOI: 10.1109/IPDPS.2011.257.

PLUGGE, E.; MEMBREY, P. & HAWKINS, A. T. 2010. *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing: 1ª Edição.* Estados Unidos da América, Apress. ISBN: 978-1-4302-3051-9

POSTGRESQL. 2014. *PostgreSQL: The world's most advanced open source database* [Online]. Disponível em: http://www.postgresql.org/ [Acedido em: 22 de Julho de 2014].

PRITCHETT, D. 2008. BASE: An Acid Alternative. *Queue,* 6**,** 48-55, ISSN: 1542-7730, DOI: 10.1145/1394127.1394128.

PROCTOR, S. 2012. *Exploring the Architecture of the NuoDB Database, Part 1* [Online]. Disponível em: http://www.infoq.com/articles/nuodb-architecture-1 [Acedido em: 7 de Abril de 2014].

PROJECT-VOLDEMORT. 2014. *Project Voldemort: A distributed database* [Online]. Disponível em: http://www.project-voldemort.com/voldemort/ [Acedido em: 3 de Abril de 2014].

REDIS. 2014. *Redis is an open source, BSD licensed, advanced key-value store* [Online]. Disponível em: http://redis.io/ [Acedido em: 3 de Abril de 2014].

REDMOND, E. & WILSON, J. R. 2012. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement: 1ª Edição.* Estados Unidos da América, Pragmatic Programmers. ISBN: 978-1-93435-692-0

RITCHIE, B. 2013. *RavenDB High Performance: 1ª Edição.* Birmingham, Inglaterra, Reino Unido, Packt Publishing. ISBN: 978-1-78216-698-6

ROBINSON, I.; WEBBER, J. & EIFREM, E. 2013. *Graph Databases: 1ª Edição.* Sebastopol, California, Estados Unidos da América, O'Reilly Media ISBN: 978-1-449-35626-2

ROUSE, M. 2014. *Definition: NuoDB* [Online]. Disponível em: http://searchdatamanagement.techtarget.com/definition/NuoDB [Acedido em: 6 de Abril de 2014].

SAGIROGLU, S. & SINANC, D. Big Data: A Review. Collaboration Technologies and Systems (CTS), 2013 International Conference on, 2013 San Diego, Califórnia, Estados Unidos da América. IEEE Computer Society, 42-47, DOI: 10.1109/CTS.2013.6567202.

SAMMER, E. 2012. *Hadoop Operations: 1ª Edição.* Sebastopol, Estados Unidos da América, O'Reilly. ISBN: 978-1-449-32705-7

SARKAR, D. 2014. *Pro Microsoft HDInsight: Hadoop on Windows: 1ª Edição.* Califórnia, Estados Undios da América, Apress Media. ISBN: 978-1-4302-6055-4

SATHI, A. 2012. *Big Data Analytics: Disruptive Technologies for Changing the Game: 1ª Edição.* Canadá, MC Press Online. ISBN: 978-1-58347-380-1

SATTAR, A.; LORENZEN, T. & NALLAMADDI, K. 2013. Incorporating NoSQL into a database course. *ACM Inroads,* 4**,** 50-53, ISSN: 2153-2184, DOI: 10.1145/2465085.2465100.

SCHMARZO, B. 2013. *Big Data: Understanding How Data Powers Big Business: 1ª Edição.* Indianapolis, Estados Unidos da América, John Wiley & Sons, Inc. ISBN: 978-1-118-73957-0

SHARMA, S. 2014. *Cassandra Design Patterns: 1ª Edição.* Birmingham, Reino Unido, Packt Publishing. ISBN: 978-1-78328-880-9

SHUTE, J.; OANCEA, M.; ELLNER, S.; HANDY, B.; ROLLINS, E.; SAMWEL, B.; VINGRALEK, R.; WHIPKEY, C.; CHEN, X.; JEGERLEHNER, B.; LITTLEfiELD, K. & TONG, P. F1 - The Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business SIGMOD/PODS '12 International Conference on Management of Data, 2012 Scottsdale, Arizona, Estados Unidos da América. ACM

SILVA, Y. N.; DIETRICH, S. W.; REED, J. M. & TSOSIE, L. M. Integrating big data into the computing curricula. Proceedings of the 45th ACM technical symposium on Computer science education, 2014 Atlanta, Georgia, Estados Unidos da América. ACM, 139-144, DOI: 10.1145/2538862.2538877.

SIMON S.Y. SHIM 2012. Guest Editor's Introduction: The CAP Theorem's Growing Impact. *Computer,* vol. 45**,** 2, DOI: 10.1109/MC.2012.54.

SINGH, S. & SINGH, N. Big Data Analytics. Communication, Information & Computing Technology (ICCICT), 2012 International Conference on, Outubro 19-20 2012 Mumbai, India. IEEE Computer Society, DOI: 10.1109/ICCICT.2012.6398180.

SMITH, T. 2013. *TED-Ed : Big Data* [Online]. Disponível em: http://ed.ted.com/lessons/exploration-on-the-big-data-frontier-tim-smith [Acedido em: 1 de Março de 2014].

SOARES, S. 2012. *A Framework that Focuses on the "Data" in Big Data Governance* [Online]. Disponível em: http://ibmdatamag.com/2012/06/a-framework-that-focuses-on-the-data-in-big-data-governance/ [Acedido em: 10 de Abril de 2014].

SOLIDIT. 2014. *DB Engines: Knowledge Base of Relational and NoSQL Database Management Systems* [Online]. Disponível em: http://db-engines.com/en/ [Acedido em: 22 de Julho de 2014].

SONG, I.-Y.; ROWEN, W.; MEDSKER, C. & EDWARD EWEN. An Analysis of Many-to-Many Relationships Between Fact and Dimension Tables in Dimensional Modeling. Proc. of the Int'l Workshop on Design and Management of Data Warehouses, 2001 Interlaken, Suíça.

STONEBRAKER, M. 2012. New opportunities for New SQL. *Commun. ACM,* 55**,** 10-11, ISSN: 0001-0782, DOI: 10.1145/2366316.2366319.

STONEBRAKER, M. & CETINTEMEL, U. "One Size Fits All": An Idea Whose Time Has Come and Gone. Proceedings of the 21st International Conference on Data Engineering, 2005 Tókio, Japão. IEEE Computer Society, 2-11, DOI: 10.1109/icde.2005.1.

STONEBRAKER, M.; MADDEN, S.; ABADI, D. J.; HARIZOPOULOS, S.; HACHEM, N. & HELLAND, P. The end of an architectural era: (it's time for a complete rewrite). Proceedings of the 33rd international conference on Very large data bases, 2007 Viena, Áustria. VLDB Endowment, DOI: 10.1109/ICDE.2005.1.

TANNIR, K. 2013. *RavenDB 2.x Beginner's Guide: 1ª Edição.* Birmingham, Inglaterra, Reino Unido, Packt Publishing. ISBN: 978-1-78328-379-8

THUSOO, A.; SARMA, J. S.; JAIN, N.; SHAO, Z.; CHAKKA, P.; ANTHONY, S.; LIU, H.; WYCKOFF, P. & MURTHY, R. 2009. Hive: a warehousing solution over a map-reduce framework. *Proc. VLDB Endow.,* 2**,** 1626-1629, ISSN: 2150-8097, DOI: 10.14778/1687553.1687609.

THUSOO, A.; SARMA, J. S.; JAIN, N.; SHAO, Z.; CHAKKA, P.; ZHANG, N.; ANTONY, S.; LIU, H. & MURTHY, R. Hive - a petabyte scale data warehouse using Hadoop. Data Engineering (ICDE), 2010 IEEE 26th International Conference on, 1-6 Março 2010 Long Beach, Califórnia, Estados Unidos da América. IEEE Computer Society, 996-1005, DOI: 10.1109/ICDE.2010.5447738.

TINKERPOP. 2014. *TinkerPop: An Open Source Graph Computing Framework* [Online]. Disponível em: http://www.tinkerpop.com/ [Acedido em: 31 de Julho de 2014].

TIWARI, S. 2011. *Professional NoSQL: 1ª Edição.* Indianapolis, Indiana, Estados Unidos da América, John Wiley & Sons. ISBN: 978-0-470-94224-6

TUDORICA, B. G. & BUCUR, C. A comparison between several NoSQL databases with comments and notes. Roedunet International Conference (RoEduNet) 10th, 23-25 Junho 2011 Laşi, Roménia. IEEE, 1-5, ISSN: 2068-1038, DOI: 10.1109/RoEduNet.2011.5993686.

ULARU, E. G.; PUICAN, F. C.; APOSTU, A. & VELICANU, M. 2012. Perspectives on Big Data and Big Data Analytics. *Database Systems Journal,* vol. III**,** 12.

VAISH, G. 2013. *Getting Started with NoSQL: 1ª Edição.* Birmingham, Reino Unido, Packt Publishing. ISBN: 978-1-84969-4-988

VENKATESH, P. 2012. *NewSQL - The New Way to Handle Big Data* [Online]. Disponível em: http://www.opensourceforu.com/2012/01/newsql-handle-big-data/ [Acedido em: 4 de Março de 2014 2014].

VICKNAIR, C.; MACIAS, M.; ZHAO, Z.; NAN, X.; CHEN, Y. & WILKINS, D. 978-1-4503-0064-3. A comparison of a graph database and a relational database: a data provenance perspective. Proceedings of the 48th Annual Southeast Regional Conference, 2010 Oxford, Mississippi, Estados Unidos da América. ACM, 1-6, DOI: 10.1145/1900008.1900067.

VINGRALEK, R.; SHUTE, J.; SAMWEL, B.; HANDY, B.; WHIPKEY, C.; ROLLINS, E.; OANCEA, M.; LITTLEfiELD, K.; MENESTRINA, D.; ELLNER, S.; CIESLEWICZ, J.; RAE, I.; STANCESCU, T. & APTE, H. F1: A Distributed SQL Database That Scales VLDB 2013 - 39th International Conference on Very Large Data Bases, 2013 Riva del Garda, Itália. ACM, DOI: 10.14778/2536222.2536232.

VOLTDB. 2014. *VoltDB is...* [Online]. Disponível em: http://voltdb.com/ [Acedido em: 7 de Abril de 2014].

VORA, M. N. Hadoop-HBase for large-scale data. Computer Science and Network Technology (ICCSNT), 2011 International Conference on, 24-26 Dezembro. 2011 Harbin, China. IEEE Computer Society, 601-605, DOI: 10.1109/ICCSNT.2011.6182030.

WANG, G. & TANG, J. The NoSQL Principles and Basic Application of Cassandra Model. 2012 International Conference on Computer Science and Service System, 2012 Nanquim, China. IEEE Computer Society, 1332-1335, DOI: 10.1109/csss.2012.336.

WARDEN, P. 2011. *Big Data Glossary: 1ª Edição.* Sebastopol, Califórnia, Estados Unidos da América, O'Reilly Media, Inc. ISBN: 978-1-449-31459-0

WEBBER, J. 978-1-4503-1563-0. A programmatic introduction to Neo4j. Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity, 2012 Tucson, Arizona, Estados Unidos da América. ACM, 217-218, DOI: 10.1145/2384716.2384777.

WEISBERG, A. & STONEBRAKER, M. The VoltDB Main Memory DBMS. In Proceedings of the IEEE Data Eng. Bul, 2013. 21-27.

WHITE, T. 2012. *Hadoop: The Definitive Guide: 3ª Edição.* Sebastopol, California, Estados Unidos da América, O'Reilly Media Inc. ISBN: 978-1-449-31152-0

WILSON, M. 2013. *Building Node Applications with MongoDB and Backbone: 1ª Edição.* Sebastopol, Califórnia, Estados Unidos da América O'Reilly Media, Inc., ISBN: 978-1-449-33739-1

ZHAO, G.; HUANG, W.; LIANG, S. & TANG, Y. Modeling MongoDB with Relational Model. Proceedings of the 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies, 2013 Xi'an, China. EEE Computer Society, 115-121, DOI: 10.1109/eidwt.2013.25.

ZIKOPOULOS, P. C.; EATON, C.; DEUTSCH, D. D. T. & LAPIS, G. 2012. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streming Data: 1ª Edição.* Estados Unidos da América, McGraw-Hill. ISBN: 978-0-07-179053-6

Anexos

Anexo A: Interfaces gráficos no sistema MongoDB

Uma outra característica que o sistema MongoDB partilha com a grande maioria dos sistemas *NoSQL* é facto de não possuir um interface visual gráfico: todas as interações com o sistema têm de ser levadas a cabo através de um interface de linha de comandos. A equipa responsável pelo sistema disponibiliza uma ferramenta *CLI* juntamente com os ficheiros do sistema, esta aplicação denomina-se de *Mongo Shell*. Os comandos de interação com o sistema introduzidos no *Mongo Shell* são em grande parte derivados da linguagem *JavaScript*. Na realidade o *Mongo Shell* não se limita a interpretar o código *JavaScript* destinado a interagir com o sistema, o mesmo é capaz de interpretar código *JavaScript* convencional. Esta funcionalidade possibilita aos utilizadores utilizarem variáveis locais e um conjunto variado de comandos/funções que assim estendem o poder e alcance das *queries* realizadas sobre o sistema. Assim, traz as vantagens de uma linguagem de programação para a linha de comandos, sem ser necessário recorrer a uma linguagem de programação independente e o respetivo compilador ou interpretador, emulando em parte o que sucede com o *SQL Server* e a linguagem *Transact-SQL*[15].

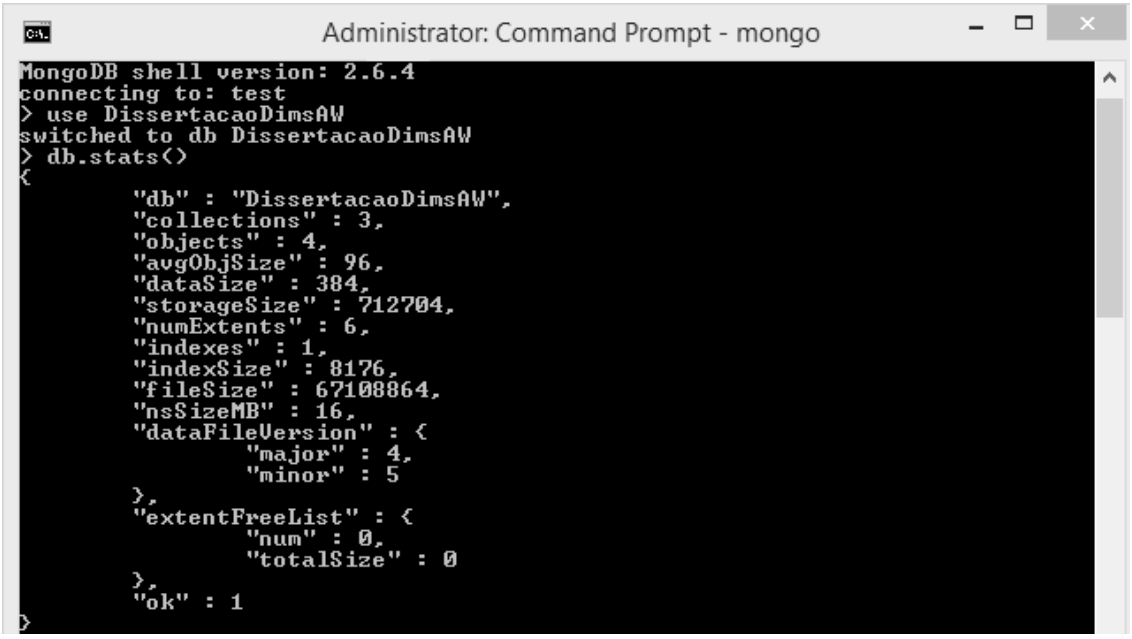

Figura 46 – Exemplo de interação com o sistema MongoDB através do MongoShell

Interagir com uma linha de comandos é em grande parte um processo nostálgico para a maioria dos "informáticos", mas a realidade é que esta situação deixa bastante a desejar em termos de usabilidade e facilidade de utilização. Felizmente o *MongoDB* possui uma comunidade bastante ativa, com vários projetos de interfaces gráficos. Apesar de vários projetos terem sido estudados e utilizados, apenas os 2 considerados favoritos serão referenciados.

[15] *T-SQL* é uma linguagem proprietária da *Microsoft* e da *Sybase*, originalmente desenvolvida pela *IBM*. A sua funcionalidade visa estender a linguagem SQL com a introdução de conceitos de programação, como variáveis locais, processamento de *strings*, dados ou funções matemáticas.

Um dos projetos selecionados foi o *Robomongo* (Paralect, 2014). Este projeto de fonte aberta e multiplataforma, ainda em versão de teste, representa uma ferramenta gráfica de gestão do sistema *MongoDB*. Esta ferramenta apresenta um visual bastante limpo e simples, que contrasta com o interface bastante complexo maioritariamente ocupado por menus e submenus presente nas outras soluções.

Um dos pontos mais interessantes deste projeto é o facto de ser praticamente uma extensão do *Mongo Shell*, na realidade o motor *JavaScript* utilizado por este projeto é exatamente o mesmo utilizada pela ferramenta CLI do MongoDB. Assim, qualquer comando ou função utilizada na Shell do MongoDB também poderá ser executada no *Robomongo*. A diferença entre os dois está no facto de que no *Robomongo* o resultado será apresentado sob uma forma gráfica, sendo que 3 formas estão disponíveis para visualizar a estrutura de documentos e coleções: vista de árvore (Figura 47), vista de tabela e vista em modo de texto (*JSON*).

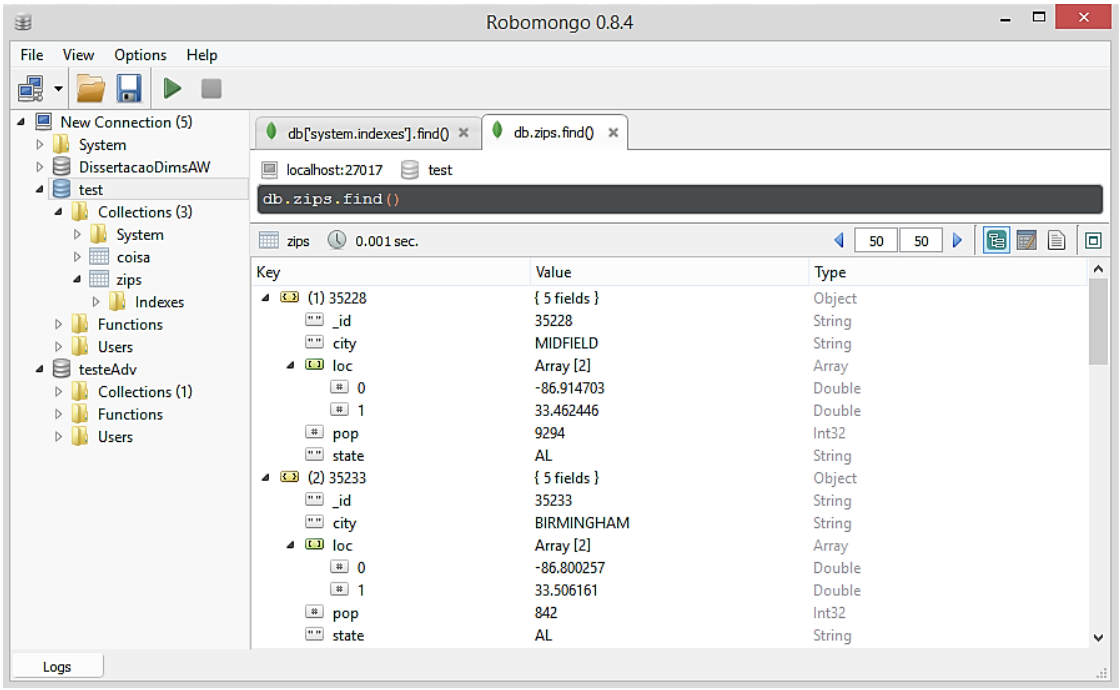

Figura 47 – *RoboMongo*, um interface gráfico de gestão do sistema *MongoDB*

O *RoboMongo* ainda fornece métodos gráficos para simples interações com o sistema, tais como criar/eliminar bases de dados, coleções ou documentos, bem como editar individualmente documentos. Este foi o método utilizado para criar a base de dados e coleção que albergam o armazém de dados no sistema MongoDB.

Infelizmente a simplicidade e facilidade de utilização do *RoboMongo*, acabam por se traduzir também no não suporte para funcionalidades mais avançadas. Devido ao facto de ainda ser uma versão de teste é de esperar que várias funcionalidades ainda sejam adicionadas antes de atingir uma versão final. Caso seja necessário recorrer a funcionalidades mais avançadas, existe um outro interface gráfico bastante interessante, *MongoVue* (MongoVUE, 2014)

O *MongoVUE* fornece todas as funcionalidades disponibilizadas pelo *RoboMongo*, a diferença está no facto de o *RoboMongo* ser visualmente mais agradável e fácil de interagir. Mas, por outro lado, o *MongoVUE* fornece algumas opções avançadas interessantes como a execução de funções *MapReduce*, cópia de coleções para outros servidores, criação de índices, gestão de utilizadores e o acesso rápido a estatísticas de coleções. Esta aplicação ainda possui algumas funcionalidades de migração de dados de bases de dados relacionais para o sistema MongoDB, e do sistema *MongoDB* para ficherios *Excel*, *JSON* ou *CSV*. Infelizmente os processos de migração de dados desta aplicação, não conseguem moldar os dados do sistema relacional para que estes encaixem no esquema MongoDB criado, assim não foi utilizado no processo de migração de dados do armazém de dados relacional para o sistema *MongoDB*. Outra questão a impedir a utilização desta aplicação foi o facto de a versão livre ser bastante limitada nas funcionalidades disponibilizadas e, após 14 dias de utilização, essas limitações ainda se tornam maiores.

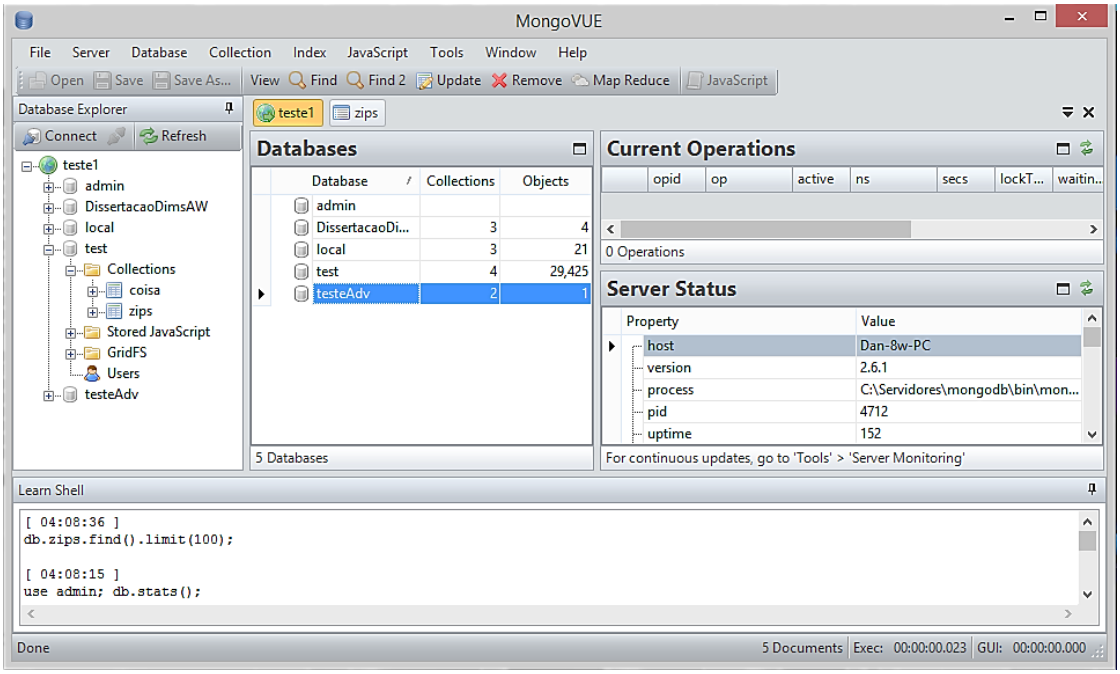

Figura 48 - *MongoVUE*, um interface gráfico mais complexo de gestão do sistema *MongoDB*

Anexo B: ETL & *OLAP* em *Big Data* e sistemas *NoSQL*

O estudo realizado no Capítulo 2.4 revelou que existiam vários processos e mudanças que precisam de ser realizadas para se conseguir integrar dados *Big Data* num armazém de dados tradicional. Outro fator salientado nesse capítulo é a questão de as análises OLAP sobre dados *Big Data* serem consideradas como algo bastante complexo de alcançar. Com este anexo pretende-se estudar e testar brevemente ferramentas que possibilitam a integração e análises de dados *Big Data*. Nesta investigação como fontes de dados *Big Data*, serão utilizadas as bases de dados *NoSQL* previamente testadas. Ao longo desta dissertação recorreu-se apenas a ferramentas cuja utilização se apresentava livre de custos para estudantes. Esta situação não se repetirá neste capítulo. Ao contrário de todos os sistemas utilizados até agora, as ferramentas utilizadas são fornecidas apenas como demonstração, estando a sua utilização limitada no tempo. Esta situação não pôde ser contornada. Simplesmente não existem outras ferramentas com condições mais favoráveis para estudantes.

Pentaho Big Data Analytics (Pentaho, 2014c)

A *Pentaho* é uma empresa norte americana especializada em soluções de *Business Intelligence (BI)*. A plataforma de *BI* desta organização possui duas versões diferentes: uma versão comunitária, de utilização livre e de fonte aberta, e uma versão comercial que possui vários componentes que estendem as funcionalidades da versão livre (Lapa et al., 2014).

Neste estudo recorreu-se à versão comercial desta plataforma conhecida como *Pentaho Business Analytics,* uma plataforma analítica bastante completa. Esta solução apresenta ferramentas e funcionalidades capazes de cobrir todas as necessidades que possam surgir no âmbito *Business Intelligence*. Esta plataforma pode ser repartida em duas categorias principais (Borland, 2014).

Pentaho Data Integration (PDI)

Este componente da plataforma fornece métodos e ferramentas capazes de integrar dados de várias fontes. A *pentaho* foi uma das primeiras organizações a orientar as soluções para tirar partido de dados *Big Data.* Desde 2010 (Borland, 2014) que esta plataforma consegue integrar e realizar consultas *OLAP* sobre estes dados.

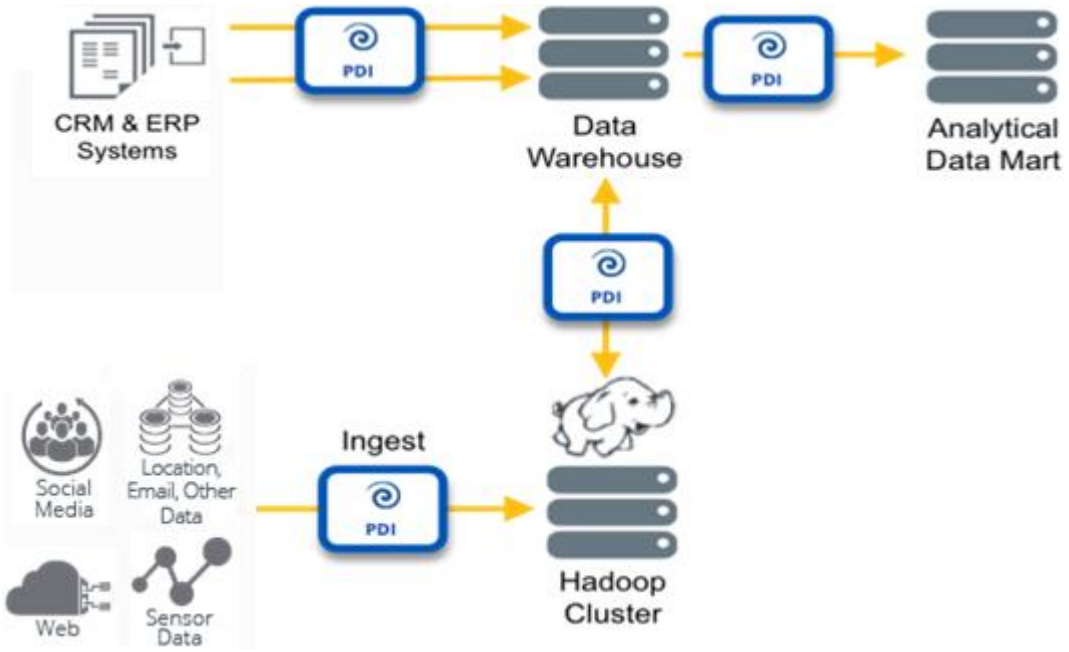

Figura 49 – Integração de dados empresariais e de dados Big Data.(Pentaho, 2014b, Pentaho, 2014a)

Na figura anterior estão ilustrados os passos que a equipa desta plataforma considera como os corretos para se integrar vários tipos de dados num armazém de dados. O ponto mais interessante a salientar da figura é o facto de que este sistema não é utilizado para tratar dados *Big Data* "crus". É sugerido que se proceda à integração desses dados em servidores *Hadoop*, para depois serem integrados no armazém de dados organizacional. A *Pentaho* não propõe uma enorme mudança no paradigma inerente aos armazéns de dados. Segundo esta filosofia, o esforço maior deve ser levado a cabo do lado dos dados *Big Data*, para que estes possam assentar corretamente no armazém de dados em causa. Como foi referido, esta plataforma possui um leque vasto de ferramentas que permitem executar várias tarefas, tais como realizar ações de *data mining*, criar relatórios interativos ou simples tabelas informativas. Essas funcionalidades são fornecidas pela segunda parte da plataforma (ainda a referir). No entanto, antes de se proceder aos vários processos de análise é normalmente necessário tratar e integrar dados de fontes distintas. Este ponto torna-se ainda mais essencial no caso de dados *Big Data.* Neste componente da plataforma referida está contida uma poderosa ferramenta de *ETL*, que foi prontamente testada.

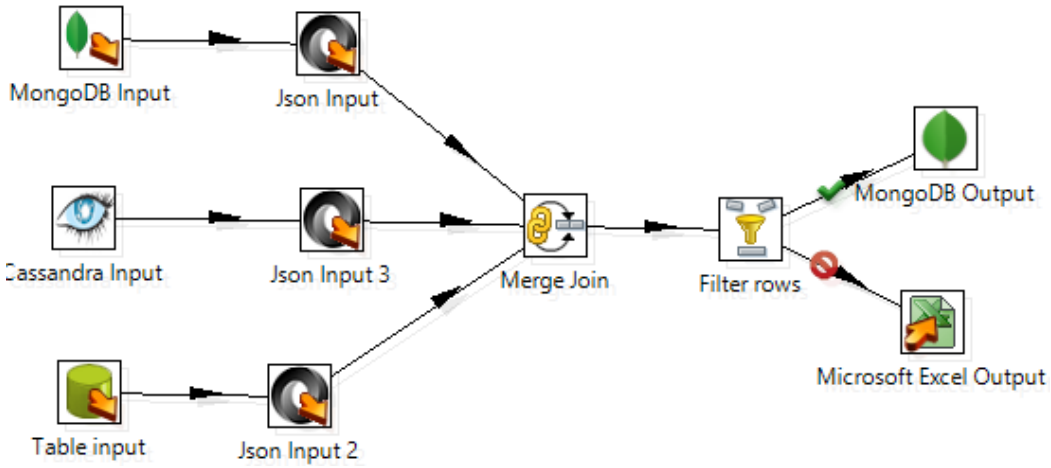

Figura 50 – Fluxo de processos desenhados na ferramenta de *ETL* da plataforma *Pentaho*

Nesta investigação não se criou nenhum repositório *Hadoop*. Deste modo, para se exemplificar e testar as capacidades desta plataforma considerou-se as bases de dados NoSQL criadas como as fontes de dados *Big Data*. Como é observável na figura anterior, criou-se um fluxo de processos que visavam integrar dados provindos do sistema *MongoDB*, *Cassandra* e do armazém de dados *SQL Server*. Este é apenas um pequeno teste, assim optou-se por filtrar todas as linhas relativas ao ano de 2008 e coloca-las no numa folha *Excel*, enquanto as restantes seriam inseridas numa nova coleção do sistema MongoDB. A capacidade que esta ferramenta apresenta para integrar dados de fontes tão heterógenas é impressionante. Esta ferramenta fornece um interface visual gráfico simples mas, ao mesmo tempo, com bastantes opções e configurações capazes de satisfazer utilizadores menos experientes e ao mesmo tempo os utilizadores avançados. Estas ferramentas de *ETL* podem ser adquiridas numa distribuição única, sem serem acompanhadas das ferramentas analíticas. O primeiro objetivo a alcançar com este foi concluído. Foi possível integrar dados *Big Data* recorrendo a uma ferramenta de *ETL*.

Pentaho Business Analytics (BA)

Este componente da plataforma inclui todas as ferramentas capazes de realizar análises de dados. Nesta solução estão incluídas todas as ferramentas *ETL* incluídas no pacote anterior. Na realidade, a conjunção destes dois componentes numa única solução torna a plataforma da *Pentaho* num verdadeiro canivete suíço do Business Intelligence. A *pentaho* foi uma das primeiras organizações a mover-se para o novo espaço aberto no mercado, para ferramentas capazes de realizar análises sobre dados *Big Data*. Um pequeno estudo sobre esta plataforma revelou que, na mesma, está incluinda uma ferramenta capaz de realizar análises *OLAP* sobre dados *Big Data*. No caso do projeto atual, como não se possui nenhum tipo de armazenamento *Hadoop*, utilizou-se a base de dados *MongoDB* criada anteriormente como fonte de dados *Big Data*, para assim se realizar algumas experiências.

Após o acesso ao sistema *MongoDB* estar configurado, o sistema irá apresentar todos os elementos presentes nos documentos armazenados na coleção *MongoDB* em causa. O sistema irá automaticamente definir quais são as medidas e os atributos dimensionais. Mas como o

sistema *MongoDB* não apresenta uma estrutura relacional, é bastante provável que este erre nesse processo (como aconteceu no exemplo prático). Felizmente é possível limpar todo o modelo e simplesmente criar um novo, bastando para esse efeito definir nomes para as dimensões e "arrastar e largar" os atributos nas dimensões pretendidas.

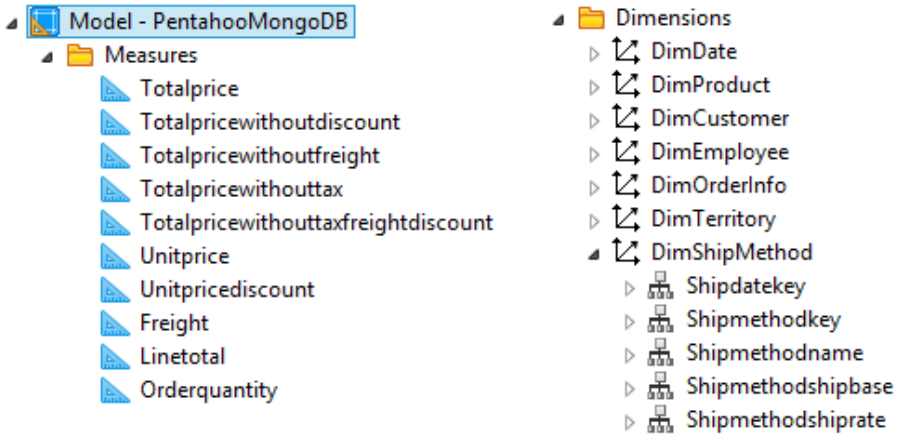

Figura 51 – Especificação de factos e medidas

Após ter-se definido o esquema ilustrado na figura anterior, foi imediatamente possível realizar análises *OLAP* sobre esses dados. No entanto, esta plataforma disponibiliza um sistema analítico em forma de servidor, o que significa que o processo de integração de dados e desenho do modelo *OLAP* pode ser completamente invisível para um elemento da organização apenas encarregue de realizar análises. Após o esquema anterior estar criado pode ser publicado no servidor analítico. Assim, os utilizadores finais que possuírem as autorizações necessárias, podem simplesmente utilizar uma aplicação web, com um design bastante interessante e intuitivo desenharem tabelas, relatórios, gráficos interativos e outra grande variedade de análises.

		2006	2007	2008	2009	2010
Customerfir	Customerlast	Orderquantity	Orderquantity	Orderquantity	Orderquantity	Orderquantity
Carla	Adams	32	32	2	-	-
Catherine	Abel	-	36	159	213	81
Frances	Adams	99	231	428	621	219
François	Ferrier	141	167	219	71	-
Gustavo	Achong	28	136	180	64	2
Humberto	Acevedo	30	34	8	-	-
Kim	Abercrombie	77	129	97	40	20
Pilar	Ackerman	-	34	213	309	135

Figura 52 – Consulta *OLAP* levada a cabo na componente analítica da plataforma *Pentaho*

A oportunidade de testar a ferramenta em causa não podia ser desperdiçada, assim, uma análise *OLAP* foi de imediato realizada, a mesma pode ser visível na figura anterior. De um modo simples foi possível obter a informação da quantidade de produtos comprados por cada cliente ao longo dos anos. Esta ferramenta ainda possibilita que sejam criados, muito facilmente, gráficos através dos relatórios e tabelas desenhados. É ainda possível combinar vários gráficos

e tabelas em *dashboards* posteriormente. Também é possível exportar esses gráficos para folhas *Excel*, documentos *PDF* ou simplesmente enviá-los para outros utilizadores.

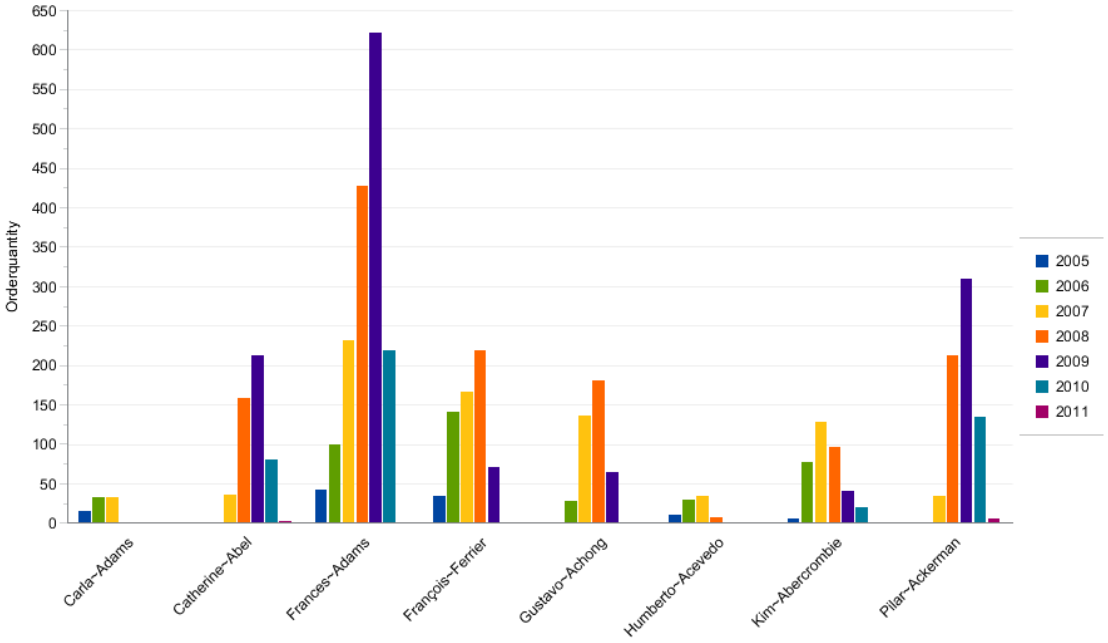

Figura 53 – Gráfico criado através dos resultados OLAP obtidos anteriormente

É importante salientar que foram utilizados muito poucos dados de modo a poder-se testar facilmente o sistema em causa. Valores mais elevados seriam obtidos, se tivesse sido utilizado o volume total de 5 milhões de linhas. Com a obtenção da tabela e gráfico anteriores considera-se que foram realizadas com sucesso análises *OLAP* sobre dados *Big Data*. Por consequência, foi assim alcançado o segundo objetivo a alcançar com a investigação deste capítulo.

Apesar de ter sido apenas testada uma plataforma, os resultados obtidos deixam uma imagem bastante positiva sobre as potencialidades deste tipo de ferramentas. Uma conclusão interessante a tirar é o facto de as plataformas de *Business Intelligence* estarem a incluir ferramentas *Big Data* juntamente com todas as outras ferramentas que tradicionalmente operam sobre fontes relacionais. A ideia passa por criar uma plataforma capaz de unificar todas as fontes de dados para que seja possível tirar proveito da combinação de diferentes tipos de dados. Na plataforma *Pentaho* testada foi utilizado um volume de dados muito baixo, não foi possível assim testar a performance dos processos de integração e análise de grandes volumes de dados. Os indícios são de que com o *hardware* correto este sistema funcionará na perfeição. A plataforma da *Pentaho* deixou uma excelente impressão e uma grande curiosidade sobre as suas verdadeiras potencialidades

Anexo C: Esquema do Armazém de Dados Relacional

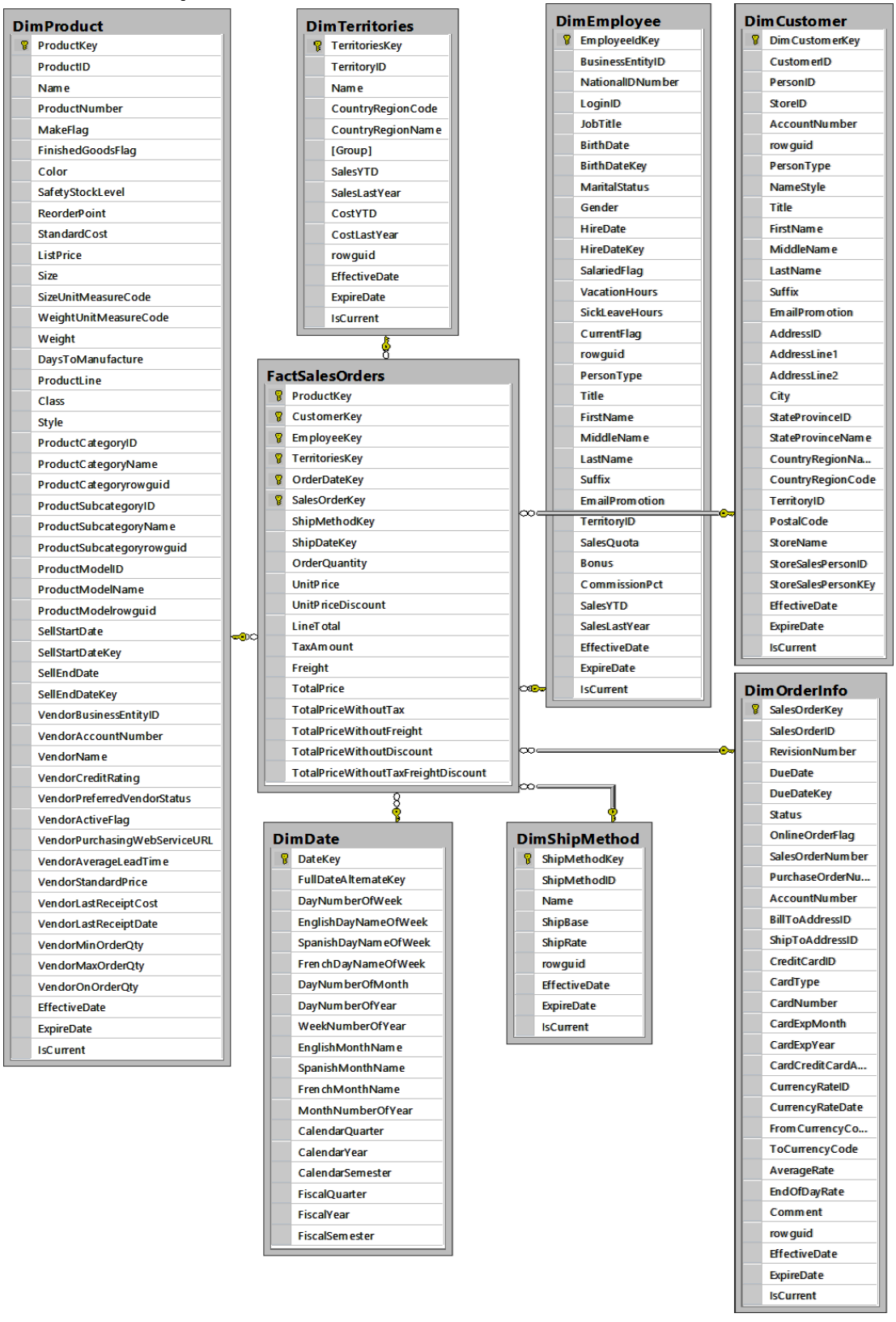

Figura 54 – Esquema em estrela (completo) ilustrativo do *armazém de dados* criado

Anexo D: Esquema do Armazém de dados MongoDB

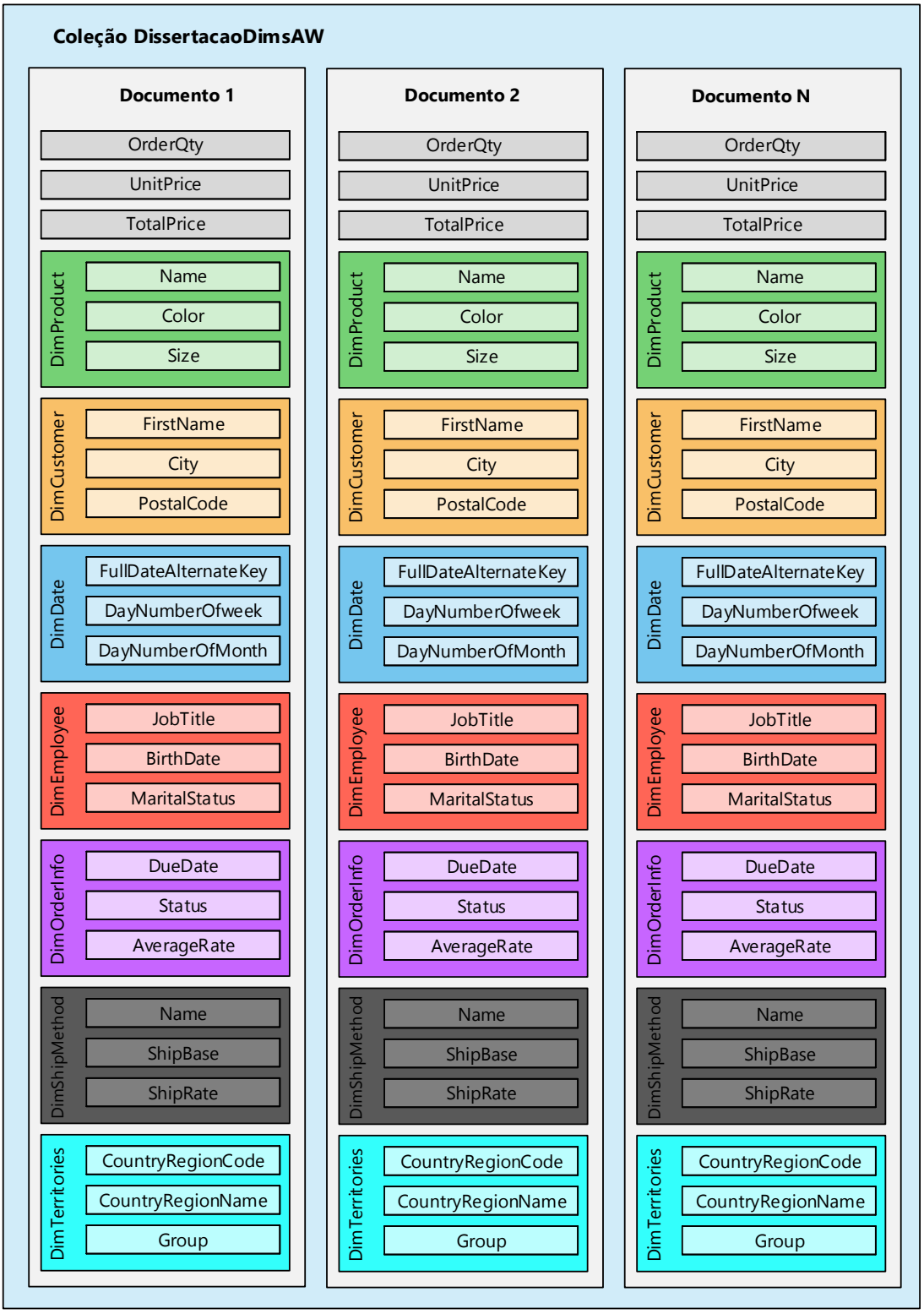

Figura 55 – Esquema (parcial) do armazém de dados relacional convertido num esquema MongoDB

Anexo E: Esquema de um documento presente no armazém de dados MongoDB


```
{
    "_id" : ObjectId("54167d4eada4761bccaa0710"),
    "ProductKey" : 1,
    "CustomerKey" : 1,
    "EmployeeKey" : 279,
    "TerritoriesKey" : 5,
    "OrderDateKey" : 20060901,
    "SalesOrderKey" : 3796,
    "ShipMethodKey" : 5,
    "ShipDateKey" : 20060908,
    "OrderQuantity" : 4,
    "UnitPrice" : 20.1865,
    "UnitPriceDiscount" : 0,
    "LineTotal" : 80.746,
    "axAmount" : 2630,
    "Freight" : 822,
    "TotalPrice" : 3532.9884,
    "TotalPriceWithoutTax" : 902.7085,
    "TotalPriceWithoutFreight" : 2711.0259,
    "TotalPriceWithoutDiscount" : 3532.9884,
    "TotalPriceWithoutTaxFreightDiscount" : 80.746,
    "DimCustomer" : {
        "CustomerID" : 29484,
        "CustomerPersonID" : 291,
        "CustomerStoreID" : 292,
        "CustomerAccountNumber" : "AW00029484",
        "CustomerRowguid" : LUUID("40124ca2-8d1a-7b42-91e9-6197f8cf3cad"),
        "CustomerPersonType" : "SC",
        "CustomerNameStyle" : false,
        "CustomerTitle" : "Mr.",
        "CustomerFirstName" : "Gustavo",
        "CustomerMiddleName" : null,
        "CustomerLastName" : "Achong",
        "CustomerSuffix" : null,
        "CustomerEmailPromotion" : 2,
        "CustomerAddressID" : 24874,
        "CustomerAddressLine1" : “3552 Mildred Ln.”,
        "CustomerAddressLine2" : null,
        "CustomerCity" : St. “Leonards”,
        "CustomerStateProvinceID" : 50,
        "CustomerStateProvinceName" : “New South Wales”,
        "CustomerCountryRegionName" : “Australia”,
        "CustomerCountryRegionCode" : “AU”,
        "CustomerTerritoryID" : 9,
        "CustomerPostalCode" : 2065,
        "CustomerStoreName" : null,
        "CustomerStoreSalesPersonID" : null,
        "CustomerStoreSalesPersonKEy" : null,
        "CustomerEffectiveD" : ISODate("2014-07-05T00:00:00.000Z"),,
        "CustomerExpireDate" : null,
        "CustomerIsCurrent" : ”Yes”
    },
    "DimTerritories" : {
        "TerritoryID" : 9,
```

```
        "TerritoryName" : "Autralia",
        "TerritoryCountryRegionCode" : "AU",
        "TerritoryCountryRegionName" : "Autralia",
        "TerritoryGroup" : "Pacific",
        "TerritorySalesYTD" : 5977814,9154,
        "TerritorySalesLastYear" : 2278548,9776,
        "TerritoryCostYTD" : 0,
        "TerritoryCostLastYear" : 0,
        "Territoryrowguid" : LUUID("5a16c46d-4c5e-d242-809d-4344e0ac75e7"),
        "TerritoryEffectiveDate" : ISODate("2014-07-05T05:22:49.000Z"),
        "TerritoryExpireDate" : null,
        "TerritoryIsCurrent" : "Yes"
    },
    "DimEmployee" : {
        "EmployeeBusinessEntityID" : 279,
        "EmployeeNationalIDNumber" : "716374314",
        "EmployeeLoginID" : "adventure-works\\tsvi0",
        "EmployeeJobTitle" : "Sales Representative",
        "EmployeeBirthDate" : ISODate("1968-02-19T00:00:00.000Z"),
        "EmployeeMaritalStatus" : "M",
        "EmployeeGender" : "M",
        "EmployeeHireDate" : ISODate("2005-07-01T00:00:00.000Z"),
        "EmployeeSalariedFlag" : true,
        "EmployeeVacationHours" : 29,
        "EmployeeSickLeaveHours" : 34,
        "EmployeeCurrentFlag" : true,
        "Employeerowguid" : LUUID("ce0f51bb-0501-0643-b591-6450d9ebf401"),
        "EmployeePersonType" : "SP",
        "EmployeeTitle" : null,
        "EmployeeFirstName" : "Tsvi",
        "EmployeeMiddleName" : "Michael",
        "EmployeeLastName" : "Reiter",
        "EmployeeSuffix" : null,
        "EmployeeEmailPromotion" : 1,
        "EmployeeTerritoryID" : 9,
        "EmployeeSalesQuota" : 300000,
        "EmployeeBonus" : 6700,
        "EmployeeCommissionPct" : 0.01,
        "EmployeeSalesYTD" : 2315185.611,
        "EmployeeSalesLastYear" : 1849640.9418,
        "EmployeeEffectiveDate" : ISODate("2014-07-05T05:22:49.000Z"),
        "EmployeeExpireDate" : null,
        "EmployeeIsCurrent" : "Yes"

    },
    "DimOrderInfo" : {
        "OrderSalesOrderID" : 47454,
        "OrderRevisionNumber" : 3,
        "OrderDueDate" : ISODate("2006-09-13T00:00:00.000Z"),
        "OrderStatus" : 5,
        "OrderOnlineOrderFlag" : false,
        "OrderSalesOrderNumber" : "SO47454",
        "OrderPurchaseOrderNumber" : "PO9570119946",
        "OrderAccountNumber" : "10-4020-000585",
        "OrderBillToAddressID" : 975,
        "OrderShipToAddressID" : 975,
        "OrderCreditCardID" : 6018,
        "OrderCardType" : “ColonialVoice”,
        "OrderCardNumber" : 77775185341217,
        "OrderCardExpMonth" : 6,
```

```
        "OrderCardExpYear" : 2018,
        "OrderCardCreditCardApprovalCode" : “717889Vi31304”,
        "OrderCurrencyRateID" : 9082,
        "OrderCurrencyRateDate" : ISODate("2007-09-01T00:00:00.000Z”),
        "OrderFromCurrencyCode" : "USD",
        "OrderToCurrencyCode" : "AUD",
        "OrderAverageRate" : 1,9294,
        "OrderEndOfDayRate" : 1,9295,
        "OrderComment" : null,
        "Orderrowguid" : LUUID("e6e95c02-c25d-3c45-8ede-53f819e80da2"),
        "OrderEffectiveDate" : ISODate("2014-07-05T05:22:49.000Z"),
        "OrderExpireDate" : null,
        "OrderIsCurrent" : "Yes",
    },
    "DimProduct" : {
        "ProductID" : 707,
        "ProductName" : "HL Road Frame - Red, 581000",
        "ProductNumber" : "FR-R92R-58-1000",
        "ProductMakeFlag" : true,
        "ProductFinishedGoodsFlag" : true,
        "ProductColor" : "Red",
        "ProductSafetyStockLevel" : 500,
        "ProductReorderPoint" : 375,
        "ProductStandardCost" : 1059.31,
        "ProductListPrice" : 1431.5,
        "ProductSize" : "58",
        "ProductSizeUnitMeasureCode" : "CM",
        "ProductWeightUnitMeasureCode" : "LB",
        "ProductWeight" : 2.24,
        "ProductDaysToManufacture" : 1,
        "ProductProductLine" : "R",
        "ProductClass" : "H ",
        "ProductStyle" : "U ",
        "ProductCategoryID" : 2,
        "ProductCategoryName" : "Components",
        "ProductCategoryrowguid" : LUUID("8d8257c6-08d8-ba4a-91a3-
af2ce02300e9"),
        "ProductSubcategoryID" : 14,
        "ProductSubcategoryName" : "Road Frames",
        "ProductSubcategoryrowguid" : LUUID("57f81555-5b07-9a4f-87b7-
43b4997077b3"),
        "ProductModelID" : 6,
        "ProductModelName" : "HL Road Frame",
        "ProductModelrowguid" : LUUID("cc2e334d-b348-044e-b7e7-
227f3ac2a7ec"),
        "ProductSellStartDate" : ISODate("2002-06-01T00:00:00.000Z"),
        "ProductSellEndDate" : null,
        "ProductVendorBusinessEntityID" : 1520,
        "ProductVendorAccountNumber" : "G&KBI0001",
        "ProductVendorName" : "G & K Bicycle Corp.",
        "ProductVendorCreditRating" : 1,
        "ProductVendorPreferredVendorStatus" : true,
        "ProductVendorActiveFlag" : true,
        "ProductVendorPurchasingWebServiceURL" : null,
        "ProductVendorAverageLeadTime" : 30,
        "ProductVendorStandardPrice" : 13.25,
        "ProductVendorLastReceiptCost" : 13.0863,
        "ProductVendorLastReceiptDate" : ISODate("2008-04-
13T00:00:00.000Z"),
        "ProductVendorMinOrderQty" : 4,
```

```
            "ProductVendorMaxOrderQty" : 200,
            "ProductVendorOnOrderQty" : 25,
            "ProductEffectiveDate" : ISODate("2014-07-05T05:22:49.000Z"),
            "ProductExpireDate" : null,
            "ProductIsCurrent" : "Yes"
        },
        "DimShipMethod" : {
            "ShipMethodID" : 5,
            "ShipMethodName" : "CARGO TRANSPORT 5",
            "ShipMethodShipBase" : 8.99,
            "ShipMethodShipRate" : 1.49,
            "ShipMethodrowguid" : LUUID("9a0166b1-34b1-764e-b957-
    2b0490c610ed"),
            "ShipMethodEffectiveDate" : ISODate("2014-07-05T05:22:49.000Z"),
            "ShipMethodExpireDate" : null,
            "ShipMethodIsCurrent" : "Yes"
        },
        "DimDate" : {
            "DateKey" : 20060901,
            "FullDateAlternateKey" : ISODate("2006-09-01T00:00:00.000Z"),
            "DayNumberOfWeek" : 6,
            "EnglishDayNameOfWeek" : "Friday",
            "SpanishDayNameOfWeek" : "Viernes",
            "FrenchDayNameOfWeek" : "Vendredi",
            "DayNumberOfMonth" : 1,
            "DayNumberOfYear" : 244,
            "WeekNumberOfYear" : 35,
            "EnglishMonthName" : "September",
            "SpanishMonthName" : "Septiembre",
            "FrenchMonthName" : "Septembre",
            "MonthNumberOfYear" : 9,
            "CalendarQuarter" : 3,
            "CalendarYear" : 2006,
            "CalendarSemester" : 2,
            "FiscalQuarter" : 1,
            "FiscalYear" : 2007,
            "FiscalSemester" : 1
        }
    }
```

Anexo F: Especificações de Hardware

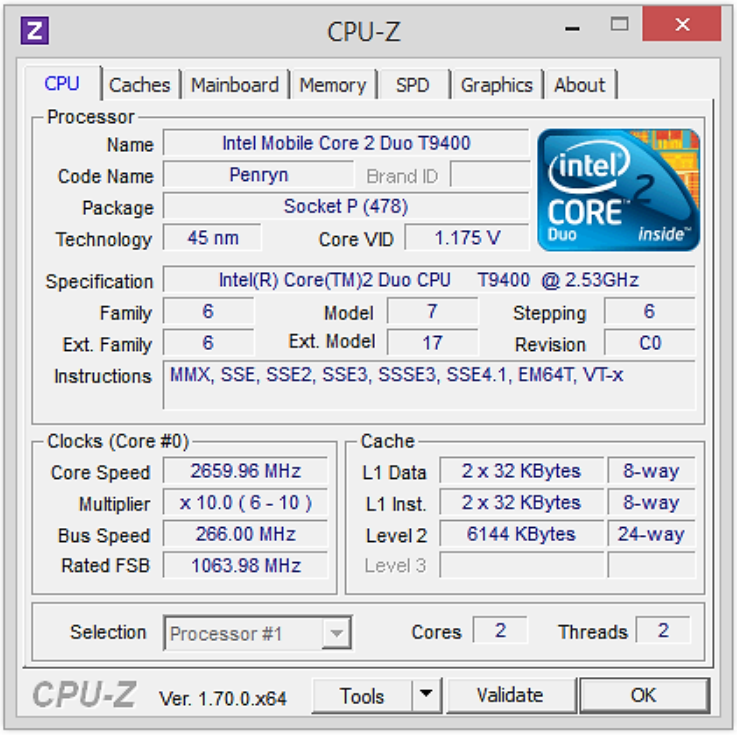

Figura 56 – Informações sobre o CPU da máquina utilizada

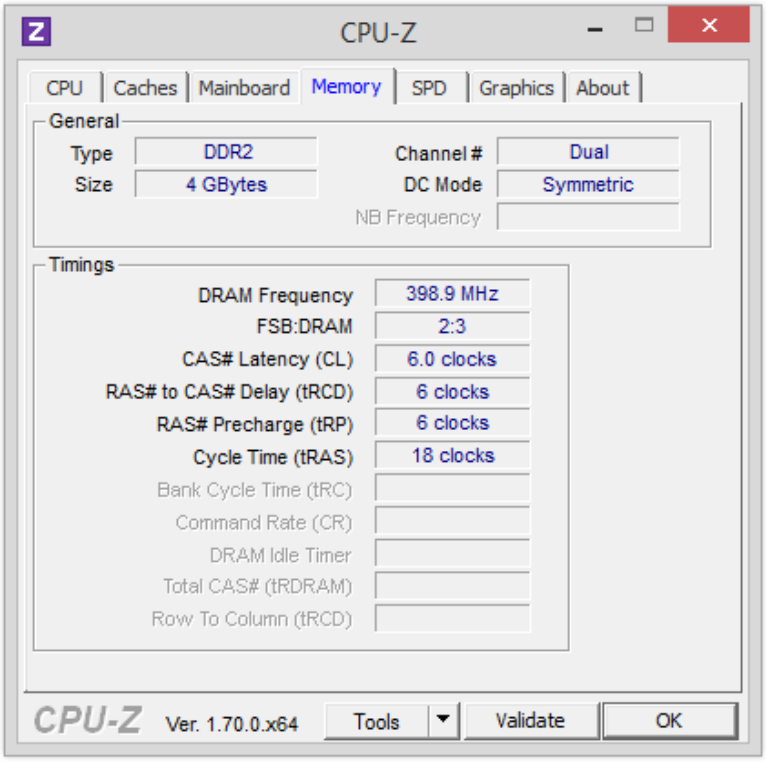

Figura 57 – Informações sobre a memória RAM disponível na máquina utilizada

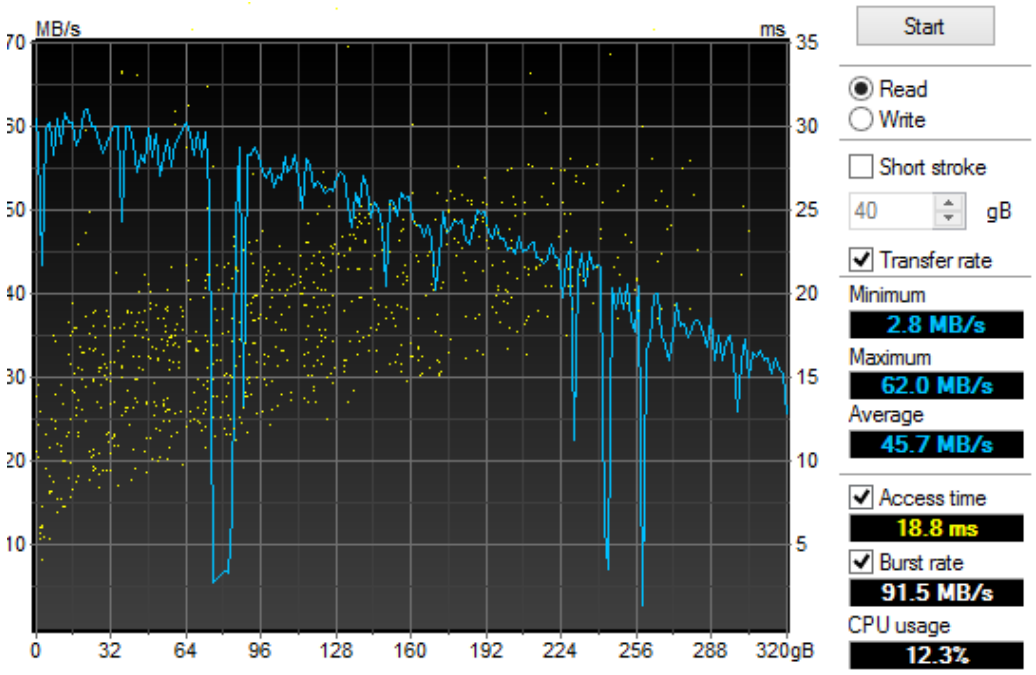

Figura 58 – Performance de leitura do disco da máquina utilizada

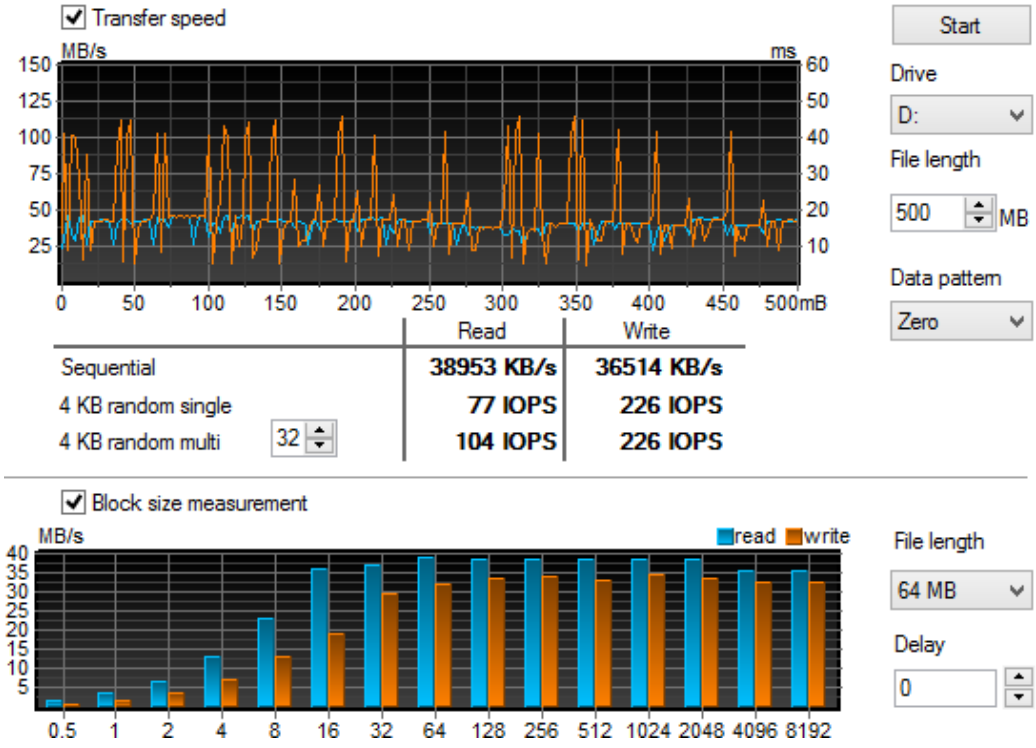

Figura 59 – Performance de transferência de ficheiros do disco rígido da máquina utilizada\