Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Tiago Alexandre Nunes Gomes

# Persistência Poliglota - Diferentes Necessidades de Armazenamento de Dados

Dezembro de 2017

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Tiago Alexandre Nunes Gomes

# Persistência Poliglota - Diferentes Necessidades de Armazenamento de Dados

**Dissertação de Mestrado**

Sistemas e Tecnologias de Informação para as Organizações

Professor Doutor Jorge Alexandre de Albuquerque Loureiro

Professora Doutora Ana Cristina Wanzeller Guedes de Lacerda

Dezembro de 2017

**Aos Meus Pais.**

*It has become appallingly obvious that our technology has exceeded our humanity*

Albert Einstein

# RESUMO

A necessidade crescente de se armazenarem grandes quantidades de dados, de forma a prover serviços escaláveis, obrigou à procura de novas soluções, ao longo do tempo. No decurso da história podem enumerar-se diversos sistemas de dados: o hierárquico, o em rede, o relacional, o orientado a objetos, o objeto-relacional e, mais recentemente, o NoSQL. Todos estes sistemas tentaram dar resposta a diferentes realidades do armazenamento de dados, indo ao encontro dos problemas de cada época.

Devido à necessidade de se aproveitarem as vantagens que cada tipo de base de dados oferece, surgiu o conceito de Persistência Poliglota, que traduz a integração de vários tipos de bases de dados num só sistema. Esta abordagem tem como objetivo retirar o melhor de cada base de dados, apresentando uma solução fiável e alternativa aos sistemas com apenas um tipo de base de dados.

Como tal, este trabalho visa a análise da abordagem de Persistência Poliglota para comparar sistemas compostos por diversos sistemas de gestão de base de dados *versus* os que utilizam apenas um motor de base de dados, de modo a verificar se esta abordagem é útil e vantajosa.

Desta forma, elaborou-se uma prova de conceito, com base num problema proposto, com o objetivo de se analisarem dois sistemas, um único e outro poliglota, tendo por base três volumes de dados diferentes. Para isto, foi necessário proceder-se à análise e escolha dos sistemas de gestão de base de dados a utilizar e montar os ambientes de teste, para ambos os sistemas. Com recurso a várias consultas individuais (a cada base de dados) e globais (conjunto das bases de dados que compõem o sistema poliglota), foram analisados os resultados obtidos com recurso à métrica de medição do desempenho relativa aos tempos de consulta.

O trabalho e os resultados obtidos evidenciaram um aumento do desempenho, quanto à utilização individual das bases de dados. Perante o conjunto das bases de dados, apesar de um ligeiro aumento, nota-se que os resultados não são claros e que carecem de uma investigação mais profunda. Por fim, é possível afirmar que a abordagem poliglota é principalmente útil em sistemas complexos, onde o volume de dados é elevado, e onde se pretende armazenar diferentes tipos de dados.

# ABSTRACT

The increasing need to store large amounts of data in order to provide scalable services has forced the search for new solutions over time. In the course of history, several data systems can be enumerated: hierarchical, network, relational, object-oriented, object-relational, and, more recently, NoSQL. All these systems tried to respond to different realities of data storage, meeting the problems of each era.

However, due to the need to take benefit from all the advantages that each type of database offers, the concept of Polyglot Persistence has emerged, which allows the integration of several types of databases in a single system. This approach aims to get the best out of each database, presenting a reliable and alternative solution to systems with only one type of database.

As such, this work aims at the analysis of Polyglot Persistence approach to compare systems composed of several database management systems versus those using a single database engine, in order to verify if this approach is useful and advantageous.

In this way, a proof of concept was elaborated, based on a proposed problem, with the objective of analyzing two systems, a single and another polyglot, based on three different data volumes. For this, it was necessary to proceed to the analysis and choice of the database management systems to be used and to assemble the test environments, for both systems. Using a number of individual queries (for each database) and global queries (set of databases that make up the polyglot system), the results obtained were analyzed using the performance metric relative to the query times.

The work and the results obtained showed an increase in the performance, regarding the individual use of the databases. In spite of a slight increase, the results are not clear and need further investigation. Finally, it is possible to affirm that the polyglot approach is mainly useful in complex systems, where the volume of data is high, and it is intended to store different types of data.

# PALAVRAS-CHAVE

Persistência Poliglota
Modelo de Dados
Bases de Dados
Sistemas de Gestão de Bases de Dados
NoSQL
Modelo Relacional

# KEY WORDS

Polyglot Persistence
Database Models
Databases
Database Management System
NoSQL
Relational Data Model

# AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer à minha família, que sempre me apoiou em todas as decisões tomadas, em especial aos meus pais, António Gomes e Maria Dulce Lopes Nunes que sempre me providenciaram todos os meios para alcançar os objetivos estabelecidos ao longo deste percurso académico. Obrigado pelo carinho e paciência.

De seguida, a ressalva ao orientador Professor Doutor Jorge Loureiro e à coorientadora Professora Doutora Cristina Lacerda, com quem pude contar de forma constante para uma boa conclusão desta etapa. Obrigado pelo apoio dado, pelos conselhos e persistência.

Agradeço ainda aos meus amigos que me acompanharam durante toda a infância e vida académica, como também aos meus colegas de mestrado e trabalho, em especial ao Nelson Marques e João Sousa, pelo companheirismo que tivemos durante esta etapa do percurso académico.

Por último e não menos importante, à minha namorada Ana Rita Pereira, pelo apoio e paciência incondicional concedida, um muito obrigado.

# ÍNDICE GERAL

# ÍNDICE DE FIGURAS

# ABREVIATURAS E SIGLAS

| | |
|---|---|
| **ACID** | Atomicity, Consitency, Isolation, Durability |
| **ADN** | Ácido desoxirribonucleico |
| **ANSI** | American National Standards Institute |
| **API** | Application Programming Interface |
| **BASE** | Basically Available, Soft State, Eventual Consistency |
| **BD** | Base de Dados |
| **BSON** | Binary JavaScript Object Notation |
| **CAP** | Consistency, Availability, Partition Tolerance |
| **CPU** | Central Processing Unit |
| **CQL** | Cassandra Query Language |
| **CRUD** | Create, Read, Update, Delete |
| **DCL** | Data Control Language |
| **DDL** | Data Definition Language |
| **DML** | Data Manipulation Language |
| **GPS** | Global Positioning System |
| **HDD** | Hard Disk Drive |
| **ISO** | International Organization for Standardization |
| **IMS** | Information Management System |
| **JSON** | JavaScript Object Notation |
| **MVC** | Model-View-Controller |
| **MVCC** | Multiversion Concurrency Control |
| **NoSQL** | Not Only Structured Query Languange |
| **OLTP** | Online Transaction Processing |
| **OO** | Orientado a Objetos |
| **OR** | Objeto-Relacional |
| **PaaS** | Platform as a Service |
| **RAD** | Rapid Application Delievery |
| **RAM** | Random Access Memory |
| **REST** | Representational State Transfer |
| **SGBD** | Sistema de Gestão de Base de Dados |
| **SGBDR** | Sistema de Gestão de Base de Dados Relacional |
| **SOAP** | Simple Object Access Protocol |
| **SQL** | Structured Query Language |
| **SSD** | Solid State Drive |
| **XML** | eXtensible Markup Language |

# 1. Introdução

Desde os vírus até aos seres mais evoluídos que a informação armazenada no código genético é passada de geração em geração, através do ácido desoxirribonucleico (ADN), de forma a garantir a evolução de cada espécie. Segundo Bill Gates, "*DNA is like a computer program but far, far more advanced than any software we've ever created*" (Rhodes, 2004). É aqui, que, sem se dar conta, está registada a nossa personalidade, características e sentimentos que, até hoje, não se conseguem replicar num simples *software*. Como tal, o genoma evidencia a importância do armazenamento da informação.

A necessidade de se guardar informação, através da simbologia e da escrita, é uma realidade, desde a pré-história. Inicialmente, recorrendo a inscrições lapidares, séculos mais tarde, em pergaminhos e, mais recentemente, na forma de papel. Antes da existência de computadores e bases de dados, foram criados repositórios de forma a armazenar toda a informação necessária, através de livros, artigos e documentos, depois arquivados em bibliotecas, já com a sua catalogação, incluindo, necessariamente, algum tipo de forma de pesquisa (Date, 2003). O aparecimento dos computadores alterou radicalmente a forma de armazenar dados, uma vez que foi possível reduzir os custos, espaço e a complexidade de armazenar, indexar e consultar estes dados, recorrendo a uma Base de Dados (BD) (Berg, Seymour e Goel, 2013).

Na década de 70, foi apresentado o Modelo Relacional, proposto por Edgar F. Codd, que veio revolucionar a forma como se guardavam os dados. O Modelo Relacional foi definido com base na teoria matemática dos conjuntos, que permite demonstrar as diversas capacidades de

modelação e consulta de dados (E. Codd, 1970). Esta solução foi assim de encontro às necessidades de armazenamento de grandes volumes de informação em formato digital, de forma eficiente, coesa e identificável (E. Codd, 1970), (Danielsen, 1998). Com base neste modelo, foram desenvolvidos diversos Sistemas de Gestão de Base de Dados Relacionais (SGBDR) que, através da linguagem padrão *Structured Query Language* (SQL), tornava possível a consulta e a manipulação dos dados armazenados. Entretanto, outros sistemas foram apresentados, como por exemplo o modelo de base de dados orientado a objetos, nos anos 80, e o modelo de base de dados objeto-relacional, na década de 90 (Damesha, 2015).

Na primeira década do novo milénio, com a crescente utilização da Internet, houve a necessidade de dar resposta a uma nova realidade de armazenamento. Segundo a Intel (Intel, 2013), até 2003, o mundo virtual continha apenas cinco *exabytes* de dados, crescendo ao longo do tempo, sendo estimados cerca de oito *zettabbytes* até 2015. Este aumento deve-se ao enorme volume de dados proveniente de diversas fontes como as redes sociais, vídeos, imagens, aplicações, documentos, entre outros. Neste contexto, surge o termo de *Big Data* que representa e descreve o resultado do crescimento exponencial do volume de dados (Cunha, 2015).

O armazenamento, a gestão e o processamento de dados desta dimensão na maioria, não estruturados, tornou evidente as limitações das bases de dados relacionais (Toth, 2011). Como consequência, emergiu uma nova perspetiva não relacional, denominada NoSQL. Apesar do termo não ser novo, uma vez que já tinha sido referido em 1998 por Carlo Strozzi (Sadalage e Fowler, 2012), trouxe uma nova solução face aos sistemas relacionais. Este modelo cresceu conjuntamente com empresas como a Google, Amazon, Twitter, Facebook, etc., uma vez que começaram a desenvolver os seus próprios sistemas de armazenamento de dados, abandonando o paradigma relacional, de forma a lidar com o enorme volume e tipo de dados, que o modelo relacional se mostra incapaz de acomodar (Grolinger et al., 2013), (Hecht e Jablonski, 2011). O NoSQL tem como objetivo a otimização do armazenamento e processamento de dados que pode ser resumida na escalabilidade, ausência de um esquema fixo e no aumento do desempenho da base de dados (Berg, Seymour e Goel, 2013).

Atualmente, muitas aplicações e sistemas têm a necessidade de lidar com dados de diversos tipos. Desta forma, é também ideal que uma aplicação recorra a bases de dados correspondentes a cada tipo, de forma a facilitar o armazenamento e processamento dos dados. Esta abordagem é já aplicada em programação, onde se utiliza um conjunto de linguagens com o propósito de

facilitar a solução de um problema ou diminuir o tempo de implementação do mesmo (Keznikl et al., 2011).

Perante uma análise profunda e ponderada da natureza dos dados de um sistema complexo, conclui-se que apenas um tipo de base de dados poderá não satisfazer os requisitos do utilizador, quanto ao armazenamento de dados. Deste modo, a solução passará pela adoção de um sistema composto por diversos tipos de base de dados, com o objetivo de dar resposta aos diferentes problemas que um sistema enfrenta. Esta solução é denominada por Persistência Poliglota (Wiese, 2015).

## 1.1 Objetivos e Motivação

Motivado pela constante evolução dos modelos de dados, pela crescente procura de soluções de base de dados para resolução de problemas complexos e por um gosto pessoal na área de bases de dados, pretende-se estudar uma nova abordagem nesta área. O conceito de Persistência Poliglota apesar de complexo, apresenta um fundamento teórico interessante. Desta forma, pretende-se estudar se esta abordagem é útil e vantajosa, analisando se constitui uma alternativa pertinente ao nível das bases de dados face às atuais.

Os requisitos das organizações que implicam lidar com dados de tipos complexos e muitos não estruturados, em volumes recentemente impensáveis, implicam a adoção de diversas abordagens. Desta forma, a adoção de um sistema composto por diferentes motores de base de dados poderá ser uma alternativa a ter em consideração.

Com este trabalho de investigação pretende-se explorar o domínio da Persistência Poliglota. Assim, procura-se atingir o seguinte conjunto de objetivos:

- Apresentar, caracterizar e distinguir os diferentes sistemas de dados no contexto do conceito de *Big Data*;
- Propor um sistema real ou fictício, tendo por base um contexto real, representativo de um sistema poliglota, onde se inclui a análise de diversos motores de base de dados;
- Construir, testar, analisar e comparar dois sistemas que seguem abordagens diferentes, constituindo uma prova de conceito;

- Identificar as vantagens e desvantagens da abordagem poliglota, relativamente aos sistemas compostos por um único tipo de base de dados.

## 1.2 Metodologia de Trabalho

Num aspeto mais formal, esta dissertação teve como metodologia de investigação a investigação-ação. Esta metodologia apresenta um duplo objetivo, de investigação e ação, de modo a se obterem resultados em ambas as vertentes. A vertente da investigação tem como objetivo aumentar a compreensão por parte do investigador, autor desta dissertação de mestrado, sobre os assuntos e matéria analisada. Em contrapartida, a ação permite aplicar, e apresentar os resultados obtidos com a investigação com o objetivo de inovar.

Zuber-Skerritt, por exemplo, sugere que uma metodologia investigação-ação é utilizada com o intuito de compreender e avaliar, visando melhorar a prática, a inovação e a mudança (Zuber-Skerritt, 2003). Esta abordagem, é assim descrita através de uma espiral de ciclos de autorreflexão, com o objetivo de planificar a mudança (Kemmis e McTaggart, 2005).

A metodologia ágil Scrum, será a metodologia de trabalho utilizada, em termos de operacionalização do processo. Esta é frequentemente utilizada no desenvolvimento de *software*, mas que, no entanto, também se adapta corretamente ao processo de elaboração de dissertações (Alipui et al., 2014).

Esta metodologia foi escolhida tendo por base os princípios de entregas constantes e incrementais de um produto final. Como é possível visualizar na Figura 1-1, esta metodologia consiste na divisão de um projeto em ciclos denominados *sprints*, onde são definidos períodos de tempo, com a duração de duas a quatro semanas, para que um conjunto de atividades seja desenvolvido.

No início de cada *sprint*, é efetuada uma reunião (*sprint planning meeting*) onde o *product owner* prioriza as tarefas mais importantes, e que devem começar a ser elaboradas primeiramente. No fim, é entregue uma parte final do trabalho desenvolvido e efetuada uma retrospetiva por parte do cliente, onde é analisado o trabalho desempenhado e reorganizadas as etapas seguintes, acrescentando assim uma fração do trabalho, ao produto final.

Esta metodologia tem como objetivo avaliar, de forma iterativa e incremental, cada fase de execução da dissertação. Desta forma, é expectável que se efetuem entregas e reuniões periódicas do trabalho escrito, dentro dos períodos de tempo estipulados, aos orientadores (Alipui et al., 2014). A metodologia Scrum foi escolhida devido à sua forma de abordar as fases da dissertação, isto é, dividindo-a em pequenos pedaços. Outro motivo desta escolha, reside na familiarização do autor com a metodologia, no ambiente profissional.

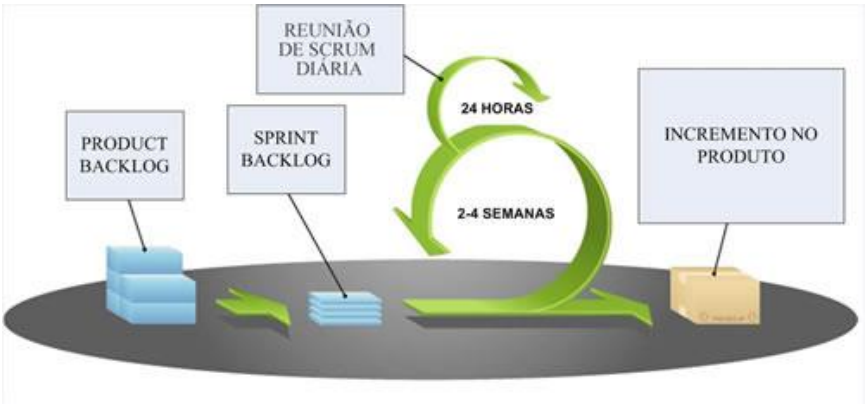



**Figura 1-1: Representação da Metodologia Scrum**
**Fonte: (Scriptcase, 2017)**

## 1.3 Estrutura da Dissertação

Esta dissertação é composta por sete capítulos principais. Estes capítulos têm como objetivo documentar os conteúdos e resultados provenientes do trabalho realizado ao longo deste projeto. Além deste capítulo, que como se viu, apresenta de forma sucinta e resumida uma introdução teórica a esta dissertação, o documento é composto por mais seis.

O capítulo dois, o Estado da Arte, consiste na contextualização do problema a tratar e na apresentação dos diversos modelos de dados. Inclui-se assim uma apresentação resumida da perspetiva histórica, características e princípios dos sistemas relacionais, NoSQL e da abordagem à Persistência Poliglota.

No terceiro capítulo, Métricas de Medição do Desempenho, serão apresentadas as métricas de medição, que se devem ter em consideração quando se prepara um servidor de base de dados de alto desempenho, assim como caso se pretenda avaliar o desempenho de um motor de base de dados.

No quarto capítulo, de seu nome Descrição do Problema, será efetuada a apresentação do problema prático, assim como da divisão entre o sistema único e o sistema poliglota. Aqui

também será apresentado o esboço do modelo conceptual de dados, assim como uma sugestão da arquitetura da aplicação.

No quinto capítulo, Sistemas de Gestão de Base de Dados Utilizados, serão apresentados os critérios de escolha dos SGBD a adotar para dar resposta ao problema descrito. Desta forma, serão apresentados, num contexto teórico, os motores de base de dados a utilizar no sistema único e nas diferentes vertentes do sistema poliglota. Aqui, tem-se como objetivo comparar, escolher e decidir qual o SGBD que melhor se adapta a cada caso a tratar.

No sexto capítulo, intitulado Prova de Conceito, será apresentado todo o trabalho prático efetuado. Aqui, está presente todo o processo, desde a instalação dos respetivos sistemas de gestão de bases de dados até à discussão dos resultados obtidos em ambos os sistemas desenvolvidos.

Por último, o sétimo capítulo, a Conclusão, terá como objetivo recolher todas as conclusões, considerações, limitações e trabalhos futuros a ter em conta, tendo por base os objetivos inicialmente traçados.

# 2. Estado da Arte

Neste capítulo será efetuada uma contextualização e revisão de um conjunto de conceitos, tecnologias e dos diversos sistemas de dados, com o intuito de enquadrar o leitor no domínio do trabalho a desenvolver.

Numa primeira perspetiva, será apresentada uma contextualização do problema no mundo real e com os requisitos cada vez mais complexos, colocados pelas organizações e clientes. Posteriormente, será apresentada uma abordagem ao fenómeno *Big Data*, um conceito atual e amplamente reconhecido como um dos principais fatores indutores da procura de uma nova solução, face aos sistemas tradicionais. No final, serão apresentadas as diferentes fases da evolução do armazenamento de dados, obviamente acompanhando a evolução temporal das exigências e das respostas encontradas, descrevendo os seus princípios e características.

## 2.1 Enquadramento

As aplicações empresariais são fundamentais na gestão de uma organização, visto que permitem facilitar a recolha, gestão e a apresentação dos dados armazenados. Uma vez que, nos dias de hoje, a informação é a chave do sucesso de uma organização (Paul, 2008), o crescimento dos dados a armazenar e a necessidade de se gerar informação a partir destes, evidenciam as debilidades das aplicações desenvolvidas. Desta forma, é imprescindível evoluir de forma paralela a este crescimento, para se dar resposta aos complexos requisitos estabelecidos pelas organizações, relativamente ao armazenamento e disponibilização da informação.

Algumas destas aplicações, apesar de já desenvolvidas, requerem uma reestruturação na sua arquitetura que dê resposta aos requisitos atuais. Isto deve-se à única realidade irrefutável: a constante mudança das arquiteturas, devido às novas realidades no armazenamento de dados. O processo de migração de base de dados nem sempre é tão trivial como se inicialmente pensa, pelo que é necessário escolher, de forma ponderada, o melhor tipo de base de dados a utilizar para cada caso em específico, na eventualidade de se optar por desenvolver um novo sistema (Zhao et al., 2014).

O volume de dados gerado pelos sistemas está a aumentar de forma desproporcional às capacidades que as bases de dados convencionais suportam. A realidade e a complexidade no armazenamento de dados, estruturados e não estruturados, do passado é totalmente diferente da atual. Desta forma, é obrigatório evoluir para soluções que apresentem um desempenho superior às tradicionais. As complicações na gestão da consistência entre os dados, a sincronização dos sistemas distribuídos por todo o globo, a gestão de falhas, a gestão da redundância dos dados, a alta disponibilidade, o desempenho dos SGBD e a segurança dos dados resumem as preocupações de uma organização, quando pondera refazer ou construir um novo sistema.

As dificuldades apresentadas traduzem as preocupações de uma organização, no que diz respeito à disponibilização da informação ao utilizador. A perda, a demora, a incorreta apresentação de dados ou a indisponibilidade de um sistema causa perdas significativas, de utilizadores e ao nível financeiro, para uma organização. Por exemplo, na Amazon, em 2008, ocorreu um episódio que se estima ter causado prejuízos de trinta mil dólares por minuto, devido a uma perda parcial do sistema (Shankland, 2008). Mais recentemente, pode-se referir o problema observado na U.S. Airlines, uma vez que se teve de cancelar cerca de dois mil voos, resultando numa perda de duzentos milhões de dólares, devido a uma falha no sistema (Ostrower, 2016).

A avaliação de uma solução, ao nível das bases de dados, é um passo fulcral para que, no futuro, esta se comporte e corresponda ao esperado. Para isto, é necessário efetuar o correto levantamento dos requisitos do sistema a desenvolver. A análise, ponderação e adoção de um SGBD adequado ao problema que se pretende resolver é o ideal para dar resposta às debilidades atualmente apresentadas, tais como a disponibilidade e o armazenamento de dados.

A componente prática desta dissertação procura avaliar se uma abordagem de Persistência Poliglota é uma valia para sistemas complexos. Pretende-se, portanto, escolher, para cada caso específico, um motor de BD que corresponda aos requisitos impostos pela organização.

## 2.2 *Big Data*

O termo *Big Data* surgiu devido à necessidade de se lidar com grandes volumes de dados gerados diariamente. Apesar de não haver uma definição consensual, o conceito foi inicialmente descrito por Doug Laney, no início dos anos 2000, baseando-se em três dimensões que se traduzem no Volume, Velocidade e Variedade dos dados, vulgarmente associado ao termo 3V's (Laney, 2001), (Kitchin, 2013), (Fan e Bifet, 2013).

A esta proposta de Laney, foram posteriormente adicionadas outras dimensões tais como, o Valor da informação gerada ao longo do tempo, a Variabilidade que se traduz nas mudanças na estrutura dos dados e, por último, a Veracidade dos dados recolhidos (Fan e Bifet, 2013), (Gandomi e Haider, 2015), (Hu et al., 2014). Das seis dimensões apresentadas, podem-se salientar as quatro mais importantes:

- Volume: A necessidade de se lidar com o crescimento exponencial de dados gerados, na ordem das dezenas e centenas de *exabytes*, armazenando-os em sistemas capazes de os gerir e disponibilizar;

- Velocidade: A necessidade de se processar os dados em tempo real, transformando-os em informação, que posteriormente é disponibilizada à organização, em tempo útil;

- Variedade: A necessidade de se guardarem diversos tipos de dados (estruturados, semiestruturados e não-estruturados) apresenta debilidades quanto ao armazenamento, dificilmente acomodável em estruturas rígidas que os sistemas relacionais disponibiliza. Estes tipos de dados podem ser definidos como texto, fotografias, ficheiros de áudio, ficheiros de vídeo, valores provenientes de sensores, documentos, e-mails, etc.;

- Valor: Necessidade de uma organização obter valor de negócio, facultado pela capacidade de resposta dos dados processados, que antes era impensável obter. Por exemplo, tenha-se em conta o caso de uma cidade inteligente, onde o objetivo é aumentar de forma sustentável a qualidade de vida, tendo por base os recursos naturais.

As dimensões apresentadas evidenciam uma nova realidade no armazenamento de dados, alterando os requisitos impostos pelos sistemas tradicionais. Estes, espelham-se principalmente na velocidade de disponibilização da informação pretendida e no armazenamento de grandes volumes de dados, de variadíssimos tipos.

O armazenamento e processamento de grandes quantidades de dados sempre foi um grande desafio, durante gerações. Ao longo do tempo, desenvolveram-se diversas soluções que acompanharam as necessidades mais prementes de cada época. Este desafio mantém-se pois, como se pode observar na Figura 2-1, além do volume de dados, acresce a variedade de tipos de dados a gravar e a tratar (dados estruturados e não estruturados) (Kitchin, 2013).

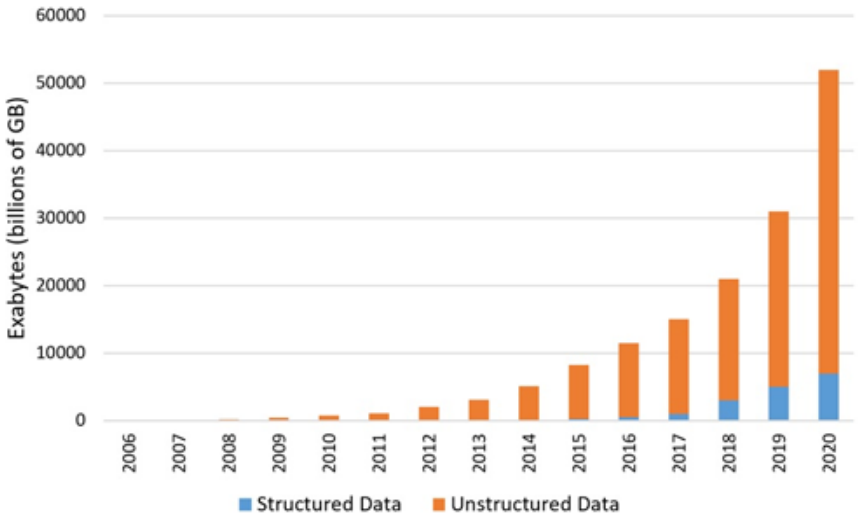



**Figura 2-1: Crescimento dos dados estruturados e não estruturados**
**Fonte: (Rizzatti, 2016)**

Os dados estruturados resumem-se a dados que possuem uma estrutura rígida. Este tipo de dados é normalmente armazenado em SGBDR, visto que têm um esquema bem definido, com campos fixos, onde o formato e o tamanho a armazenar é normalmente previsível (e.g. texto, inteiro, data, moeda, etc.). No que diz respeito aos dados semiestruturados, admite-se que possuem uma estrutura irregular ou não organizada, tal como os ficheiros *JavaScript Object Notation* (JSON) e *eXtensible Markup Language* (XML) (Coronel e Morris, 2017), (Purcell, 2012).

No campo oposto, os dados não estruturados são aqueles que não possuem uma estrutura definida (Coronel e Morris, 2017). A origem destes dados é maioritariamente proveniente de transações *online*, *emails*, vídeos, imagens, músicas, sensores, aplicações para a Internet e

*smartphones*, ficheiros, etc. (Sagiroglu e Sinanc, 2013). A quantidade elevada de tipos e dados a armazenar mostrou que os sistemas tradicionais não são os mais indicados para recolher, armazenar, gerir e analisar os dados em tempo útil (Hu et al., 2014).

Outra característica importante relativa ao *Big Data*, referida anteriormente, é o valor e a propriedade dos dados armazenados. Atualmente, pode-se afirmar que o recurso mais valioso no mundo é a informação, deixando assim para trás o petróleo e outras matérias primas. A era digital é, de certa forma, dominada por cinco empresas que lucram milhares de milhões de euros mensalmente – Alphabet, Amazon, Apple, Facebook e Microsoft. Todos os dados gerados pelos inúmeros utilizadores e clientes destas empresas são armazenados e traduzidos em informação, tornando-a valiosa (The Economist, 2017).

Através da utilização dos serviços disponibilizados pelas empresas anteriormente referidas, é possível saber o que as pessoas procuram (Google), o que partilham (Facebook), o que compram (Amazon) e o que fazem digitalmente, através da utilização de sistemas operativos (Google e Microsoft). O termo "God's Eye" poderá ser aplicado a esta realidade, uma vez que estas organizações conhecem tudo o que o utilizador faz com os seus serviços/produtos, permitindo, com base nas informações recolhidas, o desenvolvimento de aplicações monetariamente atrativas (The Economist, 2017).

Todo o armazenamento dos dados gera alguma controvérsia junto dos utilizadores, uma vez que se tratam de dados pessoais e muitas vezes confidenciais, sem que o arquivo destes seja aceite e autorizado pelo utilizador em questão (Cassandra e Petropulos, 2016).

Para enfrentar os desafios apresentados pelos conceitos associados ao *Big Data*, foram surgindo diversos sistemas de bases de dados não relacionais que, posteriormente, foram agregados noutro conceito, o NoSQL, que será abordado mais à frente.

## 2.3 Persistência Poliglota

Nas últimas décadas, como já foi atrás indicado, a quantidade de dados cresceu exponencialmente, acompanhado pelo aumento acentuado da variedade de dados. De forma a dar resposta a este facto, foram desenvolvidos ao longo do tempo soluções que permitem armazenar estes dados, conforme as respetivas necessidades.

A denominação Persistência Poliglota foi introduzida em 2008 por Scott Leberknight, e deriva de outro termo familiar, a Programação Poliglota (Leberknight, 2008), (Srivastava e Shekokar, 2016). Em 2006, Neal Ford (apud Sadalage e Fowler, 2012) apresentou o termo Programação Poliglota, para exprimir a utilização de diversos tipos de linguagens de programação, retirando de cada uma as respetivas vantagens, de forma alternativa à utilização de uma única linguagem de programação. Assim, segundo ele, era possível resolver diferentes tipos de problemas, adotando a linguagem mais adequada a cada, o que facilitaria o desenvolvimento de um sistema, com a consequente redução do tempo de implementação do mesmo (Sadalage e Fowler, 2012), (Wampler e Clark, 2010).

Tal como a Programação Poliglota, a Persistência Poliglota tem como objetivo descrever um sistema que utiliza diversos tipos de SGBD, escolhidos com base no tipo, volume e características dos dados a armazenar, retirando as vantagens possíveis de cada um. Desta forma, é possível ter uma solução enquadrada com as diferentes abordagens de persistência, para resolver os diferentes problemas apresentados por um sistema ou aplicação (Lamllari, 2013), (Wiese, 2015), (Schaarschmidt, Gessert e Ritter, 2015).

Um exemplo simples e intuitivo, onde se pode verificar a aplicação deste conceito, é frequentemente associada a uma plataforma de vendas *online* (*e-commerce*). As lojas *online* estão atualmente em grande crescimento, devido à facilidade em satisfazer o cliente através de um clique, sem que este se tenha de deslocar a uma loja física. Muitas destas lojas não estão facilmente acessíveis no local onde o cliente se encontra, pelo que, através da Internet é possível comprar os produtos mais restritos, a preços mais convidativos (Srivastava e Shekokar, 2016).

Na Figura 2-2 pode-se observar que é possível efetuar a divisão de uma plataforma ou loja de vendas *online* em quatro vertentes, em termos do funcionamento da loja em questão. A primeira vertente, a vermelho, enquadra-se no armazenamento dos dados relativos ao Carrinho de Compras e Dados de Sessão, onde se opta por adotar uma base de dados chave-valor. A segunda, a verde, Encomendas Finalizadas onde se opta por utilizar uma base de dados de documentos. Para a terceira, a azul, Inventário e Preços, opta-se por utilizar uma base de dados relacional. Por último, na quarta vertente, a cor de laranja, Partilha e Recomendação de Produtos, foi adotada uma base de dados em grafo.

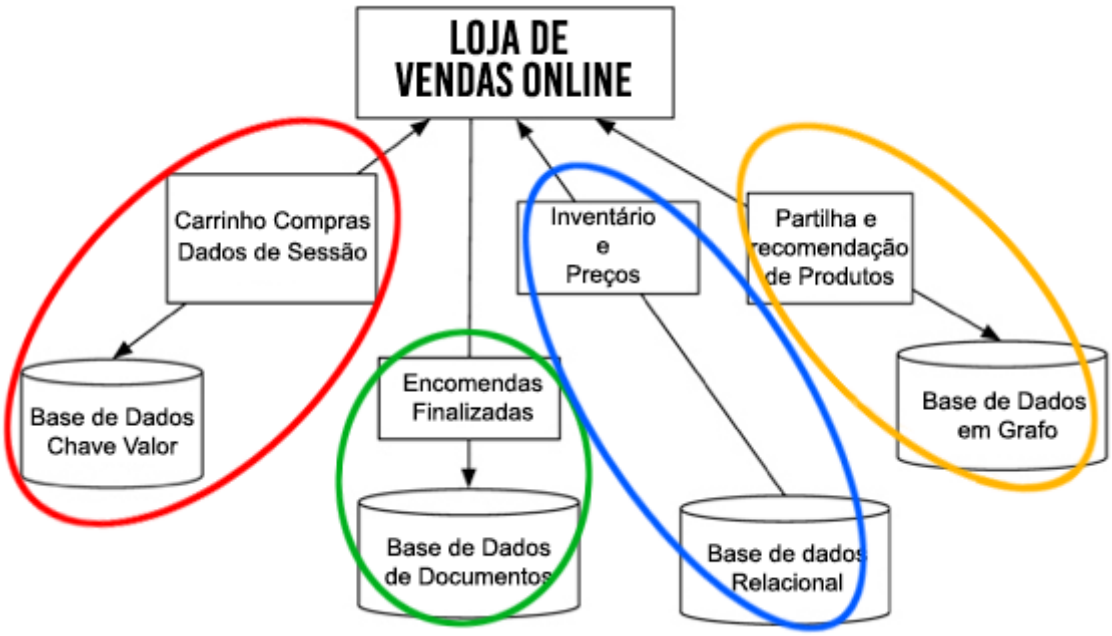



**Figura 2-2: Proposta de divisão de uma plataforma de vendas *online***
**Adaptado de: (Sadalage e Fowler, 2013)**

Num sistema complexo como este, com esta diversidade de funcionalidades, foi possível enquadrar cada vertente numa arquitetura de base de dados diferente e que melhor se adapta a cada uma das componentes apresentadas (Sadalage e Fowler, 2013).

Segundo o autor deste exemplo, esta abordagem tem como objetivo adaptar-se aos problemas propostos, apresentando a melhor solução para cada problema, como indica a figura. O esquema apresentado, apesar de possuir diversos sistemas de gestão de base de dados, apenas derivam de duas gerações de bases de dados. Veja-se que, para o componente relativo ao Inventário e Preços, é proposta uma abordagem relacional, para se armazenarem os dados, enquanto que os restantes adotam uma abordagem diferente, relativa ao NoSQL.

Estas abordagens serão referidas nas próximas secções, tendo por base as características e princípios pelas quais se regem.

## 2.4 Sistemas Relacionais

É impossível referir os modelos de dados mais atuais sem se fazer referência aos modelos de dados primordiais. Nesta secção serão apresentadas algumas formas de armazenamento relacionais mais populares, tendo como principal foco o Modelo Relacional.

### 2.4.1 Perspetiva Histórica

Até à atualidade, foram apresentados diversos modelos de dados, uns mais amplamente reconhecidos e utilizados do que outros, pois muitos, de existência fugaz, acabaram por cair em esquecimento. O modelo de dados hierárquico é considerado o primeiro modelo de dados. Desenvolvido na década de 60, tinha como objetivo gerir e armazenar dados na ordem do *megabyte*. Neste modelo, tal como o nome indica, os dados estão organizados hierarquicamente, de forma análoga a uma árvore invertida (Rob e Coronel, 2009).

A estrutura hierárquica, como se pode visualizar pela Figura 2-3, é composta por vários níveis. O primeiro nível é composto pelo elemento pai (*Root*), que posteriormente se decompõe em vários filhos, passando estes a serem pais de um número indeterminado de filhos e assim sucessivamente. Este exemplo é capaz de mostrar que este modelo se rege por relações de um para muitos (1:N), uma vez que os elementos pai podem ter diversos filhos, mas os filhos têm sempre um elemento pai. O sistema de gestão de base de dados mais reconhecido deste tipo foi o *Information Management System* (IMS), apresentado pela IBM em 1968, desenvolvido para *mainframes* (Powell, 2006), (Kedar, 2007), (IBM, 2012), (Rob e Coronel, 2009).

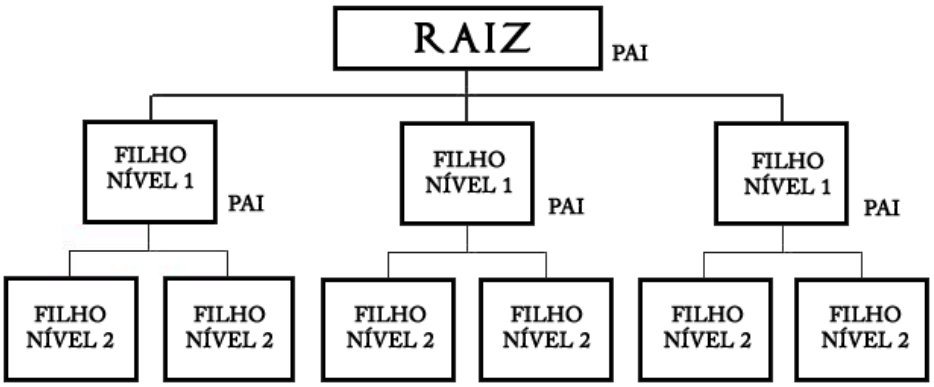



**Figura 2-3: Estrutura do Modelo de Dados Hierárquico**

Em 1969, desenvolvido por Charles Bachman, surgiu o modelo de dados em rede, como forma de melhorar o modelo de dados hierárquico (Connolly e Begg, 2005). Este modelo, com uma estrutura baseada em grafo, trouxe melhorias significativas ao modelo anteriormente apresentado, ao nível de estruturação e consulta de dados. De forma complementar ao modelo de dados hierárquico, onde o registo pai pode ter vários filhos (1:N), mas um filho apenas está relacionado com o registo pai, neste modelo, um registo poderá ter mais que um pai, o que possibilita a criação de "relacionamentos" de muitos para muitos (M:N) (Powell, 2006),

(Connolly e Begg, 2005), (Kedar, 2007). Neste contexto, o modelo em questão permite pesquisas mais rápidas e flexíveis, uma vez que podem ser elaboradas de diversas formas, enquanto que no modelo hierárquico a pesquisa tinha de ser iniciada obrigatoriamente pela raiz, descendo depois até ao nó onde se encontrava a informação pretendida.

Um ano mais tarde, em 1970, foi apresentado por Edgar F. Codd o modelo relacional, através do artigo publicado "*A Relational Model of Data for Large Shared Data Banks*" que revolucionou e ofereceu uma abordagem diferente às apresentadas até ao momento (E. Codd, 1970), (Connolly e Begg, 2005). Nesta publicação, foram apresentados vários princípios e fundamentos inovadores, sustentados e demonstrados em conceitos matemáticos (teoria dos conjuntos), permitindo depois a respetiva implementação tecnológica, suportada no modelo proposto (Connolly e Begg, 2005).

Em meados da década de 80, surgiu o modelo de dados Orientado a Objetos (OO), traçando o paralelo com a programação orientada a objetos. Tal como o nome indica, este modelo é baseado na coleção de um conjunto de objetos que contêm implicitamente informação. Este modelo surgiu devido à necessidade de se armazenarem dados complexos (ficheiros, imagens, documentos) e não homogéneos, sem suporte por parte do modelo relacional. Esta abordagem permitiu também aplicar alguns conceitos da programação orientada a objetos como as classes, objetos e métodos (Maurer et al., 1998), (Powell, 2006).

Uma década mais tarde foi desenvolvido o modelo Objeto-Relacional (OR), que teve como objetivo principal preencher a barreira existente entre o modelo de dados relacional e o orientado a objetos. Esta barreira devia-se principalmente à não existência de uma estrutura (relacional) entre os dados, visto que estes eram armazenados através de objetos e pela pouca consistência dos dados. Desta forma, o Modelo OR surgiu com a particularidade de conseguir adotar e suportar os componentes básicos que o modelo OO disponibiliza, tais como objetos, classes, etc., assim como permitiu a qualquer programador implementar novos tipos de dados, métodos e funções a utilizar na base de dados em questão. Contrariamente ao Modelo OO, este permite a consulta de dados através da linguagem declarativa SQL.

O modelo relacional é amplamente utilizado nos dias de hoje, sendo uma das maiores referencias relativas aos modelos de dados utilizados, devido à sua utilização massiva que é justificável pelos princípios pelo qual se rege (Kamal e Fouzia, 2017)

### 2.4.2 Princípios e Características

O modelo proposto por Codd baseia-se na associação entre os conceitos e regras que norteiam a teoria matemática dos conjuntos, uma vez que garantem um método formal e rigoroso para a modelação dos dados.

Resumidamente, uma base de dados relacional é composta por um conjunto de tabelas bidimensionais, chamadas relações. Uma tabela é composta por um número fixo de colunas, chamados de atributos e um número indeterminado de linhas chamadas de tuplos. Cada coluna corresponde a um atributo que contém um tipo de dados, disponibilizado pelo sistema de gestão de base de dados (Connolly e Begg, 2005). Na Figura 2-4, pode-se visualizar a representação gráfica da tabela *newsletter_regs*, composta por quatro colunas (*id*, *first_name*, *last_name*, *email*) e por seis linhas com dados correspondentes às pessoas a quem será enviada a *newsletter*.


| id | first_name | last_name | email |
|---|---|---|---|
| 1 | Armando | Roggio | armando@some.com |
| 2 | Barack | Obama | barack@usa.com |
| 3 | Bono | Vox | bono@u2.com |
| 4 | Bill | Gates | bill@microsoft.com |
| 5 | Steve | Jobs | steve@apple.com |
| 6 | Larry | Ellison | larry@oracle.com |

**Figura 2-4: Exemplo explicativo de uma tabela do modelo relacional**

Uma linha, poderá não conter dados em todas as colunas, isto é, podem existir atributos com valores nulos. Para se elaborarem consultas aos dados é geralmente utilizada a linguagem de consulta estruturada (*Structured Query Language*). Relativamente à relação, poderá ser caracterizada através do grau e da cardinalidade. O grau corresponde ao número de atributos, enquanto que a cardinalidade corresponde ao número de tuplos de uma tabela. Apesar de o grau não ser comummente variável, a cardinalidade é, devido à constante inserção e remoção de dados (Kedar, 2007).

Cada linha relativa a uma tabela é identificável univocamente através de um identificador único, chamada chave primária, que se caracteriza como não redundante e não nula, garantindo assim a unicidade de cada linha. Com isto, é possível relacionar as tabelas através de chaves estrangeiras que permitem estabelecer ligações lógicas entre linhas de tabelas. As chaves estrangeiras são colunas (uma ou mais), que representam o valor da chave primária da tabela a

relacional. Estes relacionamentos podem ser definidos como relacionamentos um para um (1:1) e um para muitos (1:N) (Powell, 2006), (Rob e Coronel, 2009).

A integridade dos dados é uma característica fundamental no modelo relacional, que poderá ser dividida em três classes: a integridade de entidade, a integridade de domínio e a integridade referencial. A primeira define que uma linha deve ser exclusiva de determinada tabela, resultando na chave primária onde os valores preenchidos são diferentes de qualquer outro. A integridade de domínio, valida se as entradas de valores para uma coluna em específico são válidas e pertencentes ao seu domínio, ou seja, se o determinado valor respeita o formato da coluna ou pertence ao intervalo de valores possíveis. Por último, a integridade referencial indica que uma chave estrangeira deverá sempre referenciar uma chave primária existente noutra tabela (Powell, 2006), (Silberschatz, Korth e Sudarshan, 2011).

O crescimento de dados resulta na degradação do desempenho do sistema. Para resolver este problema nos sistemas tradicionais, é utilizada, na maior parte dos casos, a escalabilidade vertical (Mohamed, Altrafi e Ismail, 2014). A escalabilidade é uma característica fundamental nas bases de dados, visto que se define pela capacidade de um sistema suportar um aumento de carga substancial, que por norma leva à degradação do seu desempenho, ao processar grandes volumes de dados (Hill, 1990). Este tipo de escalabilidade (vertical) consiste, principalmente, em aumentar o poder do servidor onde está armazenada a base de dados, com a adição de recursos ao nível do *hardware*, que se traduz num enorme investimento (Tiwari, 2011), (Mohamed, Altrafi e Ismail, 2014). Estes recursos visam assim aumentar a capacidade de armazenamento e processamento dos dados.

### 2.4.3 Sistemas de Gestão de Base de Dados Relacionais

Um sistema de gestão de base de dados relacional consiste num sistema, ferramenta ou aplicação que permite a criação e gestão de bases de dados. Para se obter um sistema de gestão de base de dados relacional pleno, Edgar F. Codd publicou um artigo em 1985, dividido em duas partes, com uma listagem de 12 regras que visam determinar e definir um SGBDR. A listagem das regras e a sua respetiva explicação estão presentes no Anexo A (E. F. Codd, 1985a), (E. F. Codd, 1985b).

O objetivo deste sistema é permitir aos seus utilizadores criar, modelar, organizar, definir e manter um modelo de dados, assim como disponibilizar diversas formas de acesso controladas

à base de dados correspondente. O SGBD permite assim, prover informação a uma *interface*, localizada entre a base de dados e os utilizadores ou aplicações, assegurando que os dados estão organizados e acessíveis (Connolly e Begg, 2005). Este *software* facilita ainda o armazenamento e acessos aos dados por parte de diversos utilizadores e/ou aplicações, disponibilizando funcionalidades e meios para se conseguir gerir a respetiva base de dados. Desta forma, retira-se às aplicações a responsabilidade da gestão dos dados, sendo unicamente efetuada pelo sistema de gestão de bases de dados (Elmasri e Navathe, 2011) .

Estes sistemas são bastante utilizados devido às suas funções, princípios e características, extremamente importantes quando se lida com informação. As propriedades fundamentais que um SGBD apresenta quando uma transação é efetuada, de forma a garantir a integridade dos dados da base de dados em questão, são conhecidas pelo acrónimo ACID, que resulta das primeiras letras das seguintes propriedades (Rob e Coronel, 2009), (Elmasri e Navathe, 2011):

- A **Atomicidade** é apresentada como uma propriedade "tudo ou nada", que garante que as transações sejam indivisíveis. Isto significa que uma transação é totalmente efetuada ou então, no caso de ocorrer algum problema durante a transação, nenhuma parte da operação é executada e que, consequentemente, a transação é desfeita;

- A **Consistência** é uma propriedade do modelo que assegura que uma base de dados permaneça consistente, quer no início como no fim da transação, preservando a consistência da base de dados. Caso uma transação seja abortada, o sistema volta ao estado anterior, à execução da transação em questão;

- O **Isolamento** garante que uma transação é efetuada de forma independente e sem qualquer interferência por parte de outra, quando executadas de forma concorrente. Isto é, quando duas ou mais transações estão a ser executadas simultaneamente, elas são serializadas e efetuadas apenas quando a primeira terminar.

- A **Durabilidade** permite que as alterações ocorridas, como resultado das transações processadas, sejam permanentes e persistentes, encarregando-se o sistema de assegurar que a transação é processada e disponibilizada com sucesso, mesmo que hajam falhas no sistema em questão.

Os SGBD ainda disponibilizam outras funcionalidades essenciais como a proteção da base de dados contra acessos não autorizados, através de serviços de autorização, o controlo de concorrência nos acessos à base de dados por parte de uma ou mais aplicações e a recuperação de falhas. A recuperação de falhas resume-se à restauração da integridade da base de dados depois da ocorrência de uma falha, através de um mecanismo de *backup* capaz de voltar a um estado onde a integridade dos dados está presente.

Muitos são os sistemas de gestão de base de dados relacionais que foram desenvolvidos ao longo do tempo. Na Tabela 2.1, pode-se visualizar um *ranking* relativo aos motores de base de dados mais comuns e populares em abril de 2017. Como se pode observar, as bases de dados melhor classificadas e que se baseiam nesta arquitetura de dados são as seguintes: Oracle, MySQL, MS SQL Server, IBM DB2, Microsoft Access e o SQLite (DB-Engines, 2017).

**Tabela 2.1: *Ranking* correspondente ao mês de abril de 2017 Fonte: (DB-Engines, 2017)**

| Rank | | | DBMS | Database Model | Score | | |
|---|---|---|---|---|---|---|---|
| Apr 2017 | Mar 2017 | Apr 2016 | | | Apr 2017 | Mar 2017 | Apr 2016 |
| 1. | 1. | 1. | Oracle | Relational DBMS | 1402.00 | +2.50 | -65.54 |
| 2. | 2. | 2. | MySQL | Relational DBMS | 1364.62 | -11.46 | -5.49 |
| 3. | 3. | 3. | Microsoft SQL Server | Relational DBMS | 1204.77 | -2.72 | +69.72 |
| 4. | 4. | ↑ 5. | PostgreSQL | Relational DBMS | 361.77 | +4.14 | +58.05 |
| 5. | 5. | ↓ 4. | MongoDB | Document store | 325.43 | -1.51 | +12.98 |
| 6. | 6. | 6. | DB2 | Relational DBMS | 186.66 | +1.74 | +2.57 |
| 7. | 7. | 7. | Microsoft Access | Relational DBMS | 128.18 | -4.76 | -3.79 |
| 8. | 8. | 8. | Cassandra | Wide column store | 126.18 | -3.01 | -3.49 |
| 9. | ↑ 10. | 9. | Redis | Key-value store | 114.36 | +1.35 | +3.12 |
| 10. | ↓ 9. | 10. | SQLite | Relational DBMS | 113.80 | -2.39 | +5.83 |

### 2.4.4 Linguagem de Consulta Estruturada

A consulta aos dados nos SGBDR é efetuada, de forma geral, através da linguagem de consulta estruturada. Desenvolvida pela IBM na década de 70, tornou-se, anos mais tarde, na linguagem padrão adotada pela *International Organization for Standardization* (ISO) e pela *American National Standads Institute* (ANSI), revolucionando a forma como se efetuava a consulta de dados (Cadcim, 2010), (Silberschatz, Korth e Sudarshan, 2011).

Esta linguagem tem como base a álgebra relacional que consiste numa forma de cálculo sobre conjuntos ou relações elaboradas através de operações que se traduzem em comandos. Estes comandos são aplicados a um conjunto de valores de entrada e transformados num único valor

de saída (Costa, 2007), isto é, a informação devolvida é organizada numa única estrutura, representada por uma tabela (Powell, 2006).

Apesar de, inicialmente, o SQL ter sido desenvolvido com o objetivo de ser uma linguagem orientada à consulta de dados foi, posteriormente, desenhada para permitir a manipulação de dados. Com isto, o conjunto de instruções que constituem a linguagem foi dividido em diversas categorias como, a *Data Definition Languange* (DDL) , a *Data Manipulation Language* (DML) e a *Data Control Language* (DCL) (Kriegel, 2011), (Silberschatz, Korth e Sudarshan, 2011).

A primeira tem como objetivo permitir a criação, alteração e eliminação de objetos de base de dados, enquanto que a segunda tem como objetivo permitir a manipulação dos dados através de comandos de consulta e a atualização de dados que são representadas pelo acrónimo CRUD – *Create* (Criar), *Read* (Consultar), *Update* (Atualizar), *Delete* (Eliminar) (Stephens, 2010), (Silberschatz, Korth e Sudarshan, 2011), (Rob e Coronel, 2009).

## 2.5 Sistemas NoSQL

Neste capítulo será abordado o NoSQL. Este acrónimo é designado para bases de dados não relacionais, isto é, que não seguem os fundamentos indicados na secção anterior. Como tal, apresenta-se assim como uma alternativa ao modelo relacional com novos princípios e caracteristicamente diferente.

### 2.5.1 Perspetiva Histórica

A nova geração de aplicações para a Internet revolucionou o conhecimento sobre a interação, partilha e desenvolvimento de aplicações. Esta revolução está associada ao termo dado por Tim O'Reilly em 2004, a Web 2.0. (Shuen, 2008), (O'Reilly, 2009).

A Web 2.0. trouxe novas necessidades no armazenamento de dados, assim como novas realidades nunca antes experienciadas. O crescimento exponencial de dados é uma realidade traduzida no conceito de *Big Data*, como já foi documentado, resultante do infindo volume de dados que as aplicações/sistemas produzem. Perante isto, foi necessário desenvolver alternativas que permitissem satisfazer as necessidades desejadas para a manipulação e armazenamento de dados que o modelo relacional de certa forma, já não oferecia (Jorgensen et al., 2014).

O termo *Not Only Structured Query Languange* (NoSQL) foi introduzido pela primeira vez em 1998, por Carlo Strozzi, para fazer referência a uma base de dados relacional. Esta base de dados tinha a particularidade de não utilizar a linguagem SQL para a consulta de dados, sendo efetuada através de *scripts* em UNIX (Abramova, Bernardino e Furtado, 2014), (Sadalage e Fowler, 2013). Foi durante a primeira década do novo milénio, que o movimento NoSQL começou a ganhar força e a ser considerado uma alternativa plausível, relativamente aos sistemas relacionais, devido à necessidade de se armazenar e processar uma quantidade enorme de dados (Tiwari, 2011).

Devido a esta nova realidade muitas empresas desenvolveram os seus próprios sistemas de bases de dados não relacionais, de forma a armazenarem os seus dados, tal como a Google e a Amazon, tornando-se as organizações pioneiras (Leavitt, 2010). A colossos Google publicou um artigo em 2003, descrevendo o seu sistema de ficheiros distribuídos denominado "Google File System", assim como o sistema de armazenamento construído para o efeito, chamado de "BigTable" (Ghemawat, Gobioff e Leung, 2003). No caso da Amazon, apresentou, em 2007, um artigo a descrever a implementação e os princípios pelos quais o seu sistema de armazenamento de dados, denominado "Dynamo", se regia (DeCandia et al., 2007).

Durante a primeira década deste milénio, foram desenvolvidos e apresentados variadíssimos sistemas, importantes para o avanço do movimento NoSQL. Apesar de ser utilizado num contexto diferente, o termo NoSQL foi novamente introduzido e discutido numa conferência organizada por Johan Oskarsson (Sadalage e Fowler, 2012). Com a conferência e a apresentação de diversos sistemas não relacionais, foi desencadeada uma procura e desenvolvimento de novos sistemas, alternativos aos sistemas relacionais (Strauch, 2011).

Apesar de algumas críticas a esta nova abordagem, acabou por ser amplamente aceite como uma tecnologia de armazenamento de dados, capaz e poderosa, como alternativa ao modelo de dados relacional, anteriormente apresentado (Toth, 2011).

### 2.5.2 Princípios e Características

O NoSQL, para ser uma abordagem tão viável, teve de aplicar princípios e implementar funcionalidades pelos quais os modelos de dados (relacionais), referidos na secção anterior, não se regem. Uma das principais características utilizada é a escalabilidade horizontal. Este tipo de escalabilidade consiste no aumento do número de máquinas disponíveis no *cluster* para que

se possa efetuar o processamento de tarefas ou serviços com recurso às diversas máquinas, como mostra a Figura 2-5, impossível de adotar nos sistemas relacionais devido à concorrência dos dados e aos princípios por quais se regem (Fowler, 2009). Desta forma, todo o processamento é distribuído pelas diversas máquinas, permitindo um melhor escalonamento dos dados.

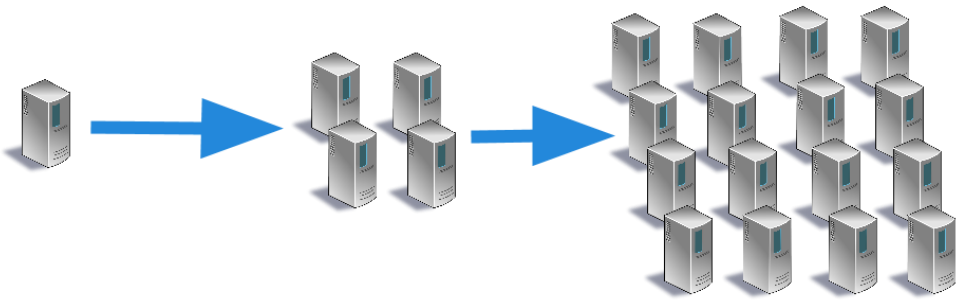


**Figura 2-5: Escalabilidade Horizontal**
**Fonte: (Voorhees, 2014)**

Em contrapartida, põe-se em causa a coerência, a redundância e a organização dos dados, uma vez que se pode obter um estado de eventual consistência, devido à sincronização dos dados pelos diversos servidores que compõem o *cluster*, como será explicado posteriormente (Sadalage e Fowler, 2012).

Eric Brewer apresentou os princípios em que assenta um sistema distribuído que, posteriormente, foram comprovados, de forma formal, por Seth Gilbert e Nancy Lynch (Brewer, 2000), (Gilbert e Lynch, 2002), (Gilbert e Lynch, 2012). Os princípios definem-se através dos três conceitos seguintes: Consistência (*Consistency*), Disponibilidade (*Availability*) e a Tolerância no Particionamento (*Partition Tolerance*) resultando no acrónimo CAP. O Teorema do CAP ou de Brewer resulta dos três princípios apresentados que se descrevem da seguinte forma:

- A **consistência** garante que, quando se efetua a atualização dos dados, correspondente a um dos nós do sistema, esta alteração será propagada aos restantes nós (Gilbert e Lynch, 2002). Durante a atualização da respetiva informação, pelos nós que compõem o *cluster*, o desempenho do sistema poderá diminuir. Isto deve-se ao elevado número de nós a atualizar, pelo que o sistema irá reunir todos os esforços para garantir a consistência dos dados em todos os nós. Desta forma, um sistema distribuído poderá adotar uma forma de "eventual consistência" (Silva, 2011);

- A **disponibilidade** assegura que um sistema permanece constantemente ativo, mesmo quando existem falhas no sistema. Desta forma, se houver algum tipo de falha em algum dos nós, os restantes deverão continuar a fornecer informação, garantindo assim a constante disponibilidade dos dados (Gilbert e Lynch, 2002);

- A **tolerância no particionamento** dos dados representa a capacidade de um sistema continuar operacional, mesmo após ser dividido. Esta divisão é efetuada quando um conjunto de nós sofre falhas, dividindo o sistema em duas partes isoladas, uma parte referente a um conjunto de nós completamente operacionais e o outro conjunto engloba os nós inoperacionais (Gilbert e Lynch, 2002).

O CAP é apresentado como um teorema, visto que os sistemas distribuídos só conseguem garantir dois, dos três conceitos descritos. Esta constatação deve-se à forma de como se relacionam as três propriedades, como se pode visualizar pela Figura 2-6 (Brewer, 2012), (Gilbert e Lynch, 2002).

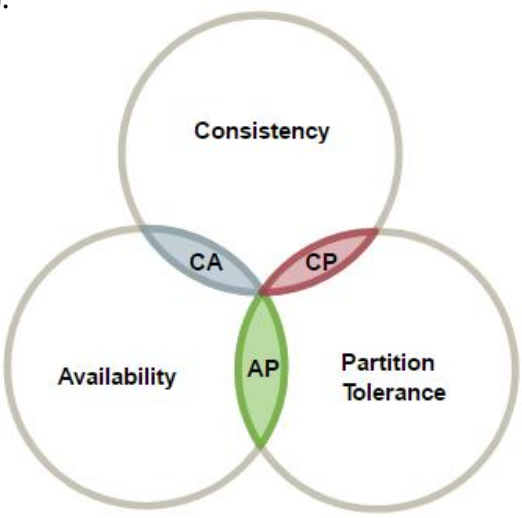



**Figura 2-6: Representação das diferentes propriedades de um sistema distribuído**
**Fonte: (Erb, 2012)**

Com isto, um sistema distribuído apenas consegue garantir duas das três propriedades em simultâneo. Desta forma, pode-se verificar a existência de três tipos de sistemas:

- Os sistemas CA possuem uma forte consistência e alta disponibilidade, sacrificando assim o princípio da tolerância na partição dos dados. Isto significa que se houver uma falha num sistema distribuído, pode-se colocar todo o sistema em risco. Deste modo, a melhor alternativa será ter-se um sistema reduzido a um nó, com alta disponibilidade, tal como os sistemas relacionais;

- Os sistemas CP possuem uma forte consistência e são tolerantes a partições, descartando o princípio da disponibilidade dos dados. Estes sistemas implicam que, quando é necessário atualizar a informação de um sistema particionado, aplicando o princípio da consistência, o sistema poderá não responder a pedidos, enquanto os dados não estiverem presentes em todos os nós do sistema;

- Quanto aos sistemas AP, baseiam-se nos princípios da disponibilidade e da tolerância à partição. Estes sistemas têm a particularidade de funcionarem em pleno, ou seja, estão sempre disponíveis para responderem a pedidos mesmo quando existem falhas no sistema. No entanto, não se garante que os dados disponibilizados ao utilizador sejam os mais atuais, só acontecendo após a sincronização dos dados pelos nós.

O princípio transacional do NoSQL é traduzido através dos seguintes conceitos: Basicamente Disponível (*Basically Available*), Estado Leve (*Soft State*) e o Eventualmente Consistente (*Eventual Consistency*), resumido pelo acrónimo BASE, que se baseia no teorema do CAP apresentado anteriormente (Gilbert e Lynch, 2002):

- O *basically available* permite ao sistema garantir a disponibilidade dos dados, havendo sempre uma resposta a qualquer pedido efetuado à base de dados;

- O *soft state* indica que o sistema não necessita de ser sempre consistente, tolerando-se assim a inconsistência dos dados por um determinado período de tempo;

- O *eventual consistency* permite ao sistema que, sempre que possível, se coloque num estado consistente, ou seja, que a informação entre os nós esteja coerente.

Com base na informação apresentada anteriormente sobre os acrónimos ACID e BASE, chega-se à conclusão que o modelo relacional tem como principal objetivo manter a consistência dos dados, contrariamente ao NoSQL que valoriza mais o desempenho da base de dados e, consequentemente, a disponibilidade dos dados (Pritchett, 2008).

Outra característica particular do modelo NoSQL é a ausência de um esquema rígido (*schema-free*), ou seja, os dados não precisam de obedecer a uma estrutura previamente definida. Esta característica é assim uma mais-valia, uma vez que a alteração do modelo de dados não implica uma possível reestruturação da base de dados utilizada.

Outra característica importante pode ser definida através do baixo custo que estas soluções apresentam perante o modelo relacional. Por exemplo, é mais barato ter-se um *cluster* composto por pequenos servidores, do que um só servidor com toda a potência agrupada, como no caso dos sistemas relacionais. Por último, o baixo custo poderá refletir-se na adoção de SGBD NoSQL abertos e disponíveis (*open-source*) (Leavitt, 2010), (Cattell, 2011).

### 2.5.3 *MapReduce*

O *MapReduc*e é uma *framework* desenvolvida inicialmente por Dean Jeffrey e Sanjay Ghemawat (Dean e Ghemawat, 2004), sendo depois adotada e introduzida pela Google em 2004. Esta *framework* tem como objetivo tratar e processar grandes volumes de dados em tempo útil, distribuindo o seu processamento por várias máquinas, ou seja, num *cluster* (Dean e Ghemawat, 2004). Apesar desta distribuição, todo o procedimento implica o processamento paralelo, uma vez que é aplicada a mesma função nas diversas máquinas relativas aos diferentes conjuntos de dados, de forma a dividir as etapas do processamento.

Esta tecnologia é ideal, no que diz respeito ao processamento intensivo de dados, visto que escala os dados por diferentes máquinas permitindo um alto desempenho. Desta forma, a sua alta escalabilidade permite processar *petabytes* de dados pelos diferentes servidores que compõem a rede (Bonnet et al., 2011). Tal como o nome indica, o *MapReduce* é um processo de duas etapas, cujos nomes são: mapear e reduzir (Dean e Ghemawat, 2004).

A primeira etapa, tem como fundamento o particionamento de um grande problema, presente num nó pai, em vários pequenos problemas que são posteriormente distribuídos pelos nós filho na rede. Na fase Reduzir estes pequenos problemas são resolvidos em cada nó filho, sendo o resultado correspondente passado ao pai (Sadalage e Fowler, 2012), (Dean e Ghemawat, 2004), (Strauch, 2011).

A melhor forma de compreender como todo este processo se desencadeia é apresentando um exemplo prático. A Figura 2-7 mostra, de forma resumida, a representação das etapas do processamento o processamento de uma lista de pares chave-valor (apresentada em (3)) de modo a criar uma nova lista, mais simples, com novos pares chave-valor (mostrada em (5)).

A função (1) demonstra o mapeamento e processamento da lista inicial, representada por (3). Esta lista será, inicialmente, mapeada por diversas máquinas, devolvendo novos pares de chave-valor obtendo-se o conjunto de valores exibido em (4). Após estar o mapeamento efetuado é

então aplicada a função de *reduce* (2) que consiste no processamento do conjunto de pares chave-valor devolvida em (4), agregando-os através de funções (SUM, AVG, etc.). Daqui resultam os novos pares chave-valor mais simples, representada pelo (5) (Bonnet et al., 2011).

$$map(k1, v1) = list(k2, v2) \quad (1)$$
$$reduce(k2, list(v2)) = list(v3) \quad (2)$$

$$List : (a;2)(a;4)(b;4)(c;5)(b;2)(a;1) \quad (3)$$
$$After\ mapping : (a;[2,4,1]),(b;[4,2]),(c[5]) \quad (4)$$
$$After\ reducing : (a;7),(b;6),(c;5) \quad (5)$$

**Figura 2-7: Exemplo prático do *MapReduce***
**Fonte: (Bonnet et al., 2011)**

Todas estas etapas estão também representadas graficamente através da Figura 2-8. A primeira etapa representada é a apresentação dos dados, ou seja, a lista representada por (3). Posteriormente, é efetuado o mapeamento de dados (1), resultando no conjunto de valores representado por (4). É então que a função de *reduce* (2) é aplicada, resultando nos valores apresentados no final deste fluxo e representada por (5).

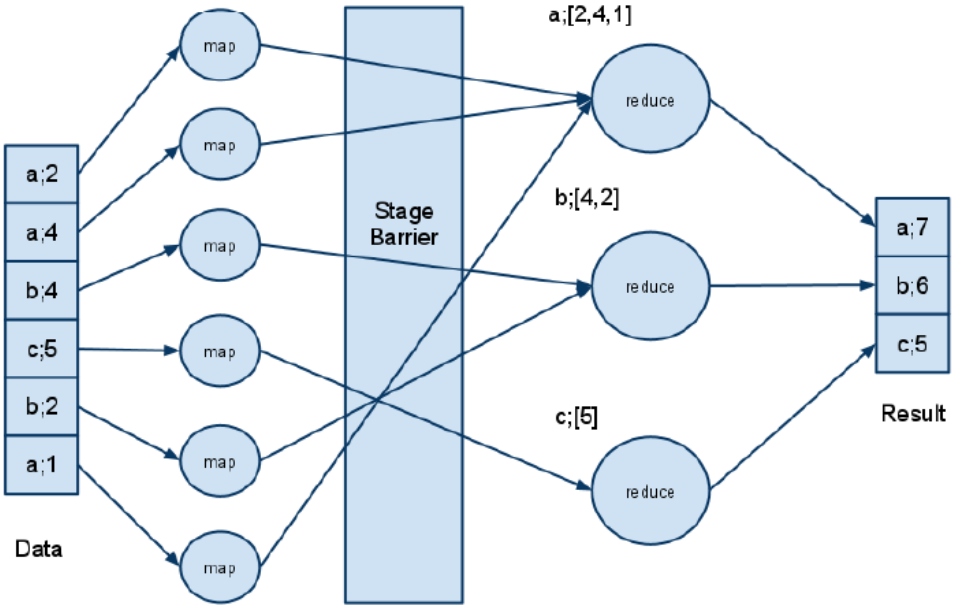



**Figura 2-8: Representação gráfica do *MapReduce***
**Fonte: (Bonnet et al., 2011)**

### 2.5.4 Sistemas de Gestão de Base de Dados NoSQL

Atualmente, existem diversos tipos de base de dados que se regem pelo modelo NoSQL, podendo ser agrupados em quatro categorias. São elas: base de dados chave-valor (*key-value*), base de dados orientada a colunas (*column oriented*), base de dados orientadas a documentos (*document stored*) e, por último, as base de dados orientadas a grafos (*graph based*) (Bonnet et al., 2011).

A base de dados por chave-valor é considerada, entre as quatro categorias apresentadas, a mais simples. Pode ser visualizada, de forma geral, como uma tabela *hash*, uma vez que as chaves são utilizadas como índices. Esta tabela pode ser interpretada e caracterizada como um conjunto de chaves que estão associadas a um valor, lista, estrutura ou objeto, obtendo assim um par chave-valor. Desta forma, o utilizador poderá efetuar consultas às bases de dados de acordo com a chave desejada (Nayak, Poriya e Poojary, 2013).

A Figura 2-9, apresenta um exemplo desta categoria, que representa o mapeamento entre o IP de uma máquina e o seu MAC Address. Como se pode observar do lado esquerdo, estão representadas chaves (IPs) e do lado direito, o valor correspondente à respetiva chave (MAC Address). Visto que a tabela é composta apenas por uma chave e o respetivo valor, este modelo confere alta disponibilidade no acesso aos dados. Tal como não poderia deixar de ser, neste tipo de base de dados as chaves são únicas, de forma a garantir a unicidade do par chave-valor.

| Key | Value |
|---|---|
| 202.45.12.34 | 01:23:36:0f:a2:33 |
| 202.45.123.4 | 00:25:33:da:4c:01 |
| 245.12.33.45 | 02:03:33:10:e2:b1 |
| 101.234.55.1 | b8:67:a3:11:23:b1 |

**Figura 2-9: Exemplo explicativo do modelo Chave-Valor**

As bases de dados tipo chave-valor têm como objetivo serem altamente escaláveis. Fornecem funcionalidades para pesquisas rápidas, assim como têm capacidades para lidar com a alta concorrência na pesquisa de dados (Nayak, Poriya e Poojary, 2013), (Hecht e Jablonski, 2011). Alguns dos sistemas de gestão de base de dados NoSQL que adotam esta abordagem são os seguintes: Amazon's Dynamo, BerkeleyDB, LevelDB, Memcached, Project Voldemort, Redis e o Riak (Atikoglu et al., 2012), (Cattell, 2011), (Sadalage e Fowler, 2013).

A segunda categoria a ser apresentada é a base de dados orientada a colunas. Pode-se classificar esta abordagem como mais complexa, que a anterior apresentada, uma vez que os dados são indexados por um tripleto contendo uma linha, colunas e o rótulo de tempo.

Apesar de apresentar uma representação gráfica bastante idêntica a uma tabela do modelo relacional, não suporta ligações entre tabelas (Hecht e Jablonski, 2011). As colunas e linhas são identificadas por chaves, enquanto que o tempo permite o versionamento dos dados.

Esta abordagem permite armazenar os dados separadamente segundo colunas, compostas pelo nome e valor de forma idêntica ao modelo chave e valor anteriormente apresentado, onde cada coluna é assim um índice da base de dados. Estas colunas são depois agregadas, sendo representadas por uma linha (Han et al., 2011).

A Figura 2-10 apresenta um exemplo representativo desta categoria. Pode-se visualizar uma família de colunas, que contém três linhas com os indentificadores representados pela *Row ID* (1, 2 e 3). Quanto às colunas, são representadas pelo *Name*, *Email* e *Website*. Neste caso, foi omitido o rótulo de tempo.

| Row ID | Columns... | | |
|---|---|---|---|
| 1 | Name | Website | |
| | Preston | www.example.com | |
| 2 | Name | Website | |
| | Julia | www.example.com | |
| 3 | Name | Email | Website |
| | Alice | example@example.com | www.example.com |

**Figura 2-10: Exemplo explicativo do modelo Orientado a Colunas**
**Fonte: (Bryden, 2017)**

Este tipo de modelo de dados NoSQL pode-se caracterizar como um sistema fortemente consistente e que permite o particionamento eficiente, uma vez que foram desenhadas para suportar o processamento paralelo em massa, num sistema distribuído (Han et al., 2011), (Sadalage e Fowler, 2013), (Hecht e Jablonski, 2011). Podem-se enumerar os seguintes sistemas de gestores de bases de dados orientadas a colunas: Cassandra desenvolvida pela rede social Facebook, BigTable desenvolvida pela Google, Amazon SimpleDB desenvolvida pela Amazon, HBase, Hypertable, etc. (Han et al., 2011).

As bases de dados orientadas a documentos, tal como o nome indica, consistem em coleções de documentos semiestruturados que podem assumir diferentes formatos, como por exemplo, XML, JSON e *Binary* JSON (BSON) (Han et al., 2011). Os documentos são representados por um identificador único, acompanhado de um objeto e um conjunto de campos de valores

simples como *strings* ou inteiros, e que estão organizados como uma base de dados chave-valor (Nayak, Poriya e Poojary, 2013), (Hecht e Jablonski, 2011), (Kaur e Rani, 2013).

Pode-se caracterizar esta abordagem como de "esquema livre" (*schema-free*) permitindo ao utilizador adicionar campos, sem afetar os restantes documentos presentes na base de dados (Leavitt, 2010). Esta abordagem também tem a capacidade de diminuir a complexidade dos dados, gerir dados estruturados, semiestruturados e não-estruturados, resultando numa categoria de alta escalabilidade, devido à ausência de um esquema definido, como foi anteriormente referido (Nayak, Poriya e Poojary, 2013), (Kaur e Rani, 2013). As bases de dados mais comuns que adotam esta abordagem são o MongoDB e a Apache CouchDB (Strauch, 2011), (Han et al., 2011).

Por último, são apresentadas as bases de dados baseadas em grafos. A sua estrutura é composta por nós (vértices do grafo) que estão relacionados através de um número indeterminado de arestas, sempre associados a um nome e, que contêm um nó direção e um nó origem. Os nós e relacionamentos contém propriedades (atributos) (Sadalage e Fowler, 2013), (Robinson, Webber e Eifrem, 2015) .

Este modelo é muito flexível, exibe um grande desempenho, no que diz respeito à consulta de dados devido à rede de nós, e apresenta numa estrutura sem um esquema definido. Contudo, apresenta uma difícil implementação, dado que implica conhecimentos matemáticos relativos à estruturação em grafo (Nayak, Poriya e Poojary, 2013), (Sadalage e Fowler, 2013), (Toth, 2011).

Como se pode verificar através da Figura 2-11, este tipo de base de dados está totalmente relacionado entre si, através das arestas que ligam os diversos nós. Neste exemplo, cada nó é representativo de uma pessoa, empresa ou objeto, enquanto que os relacionamentos mostram a ligação entre eles.

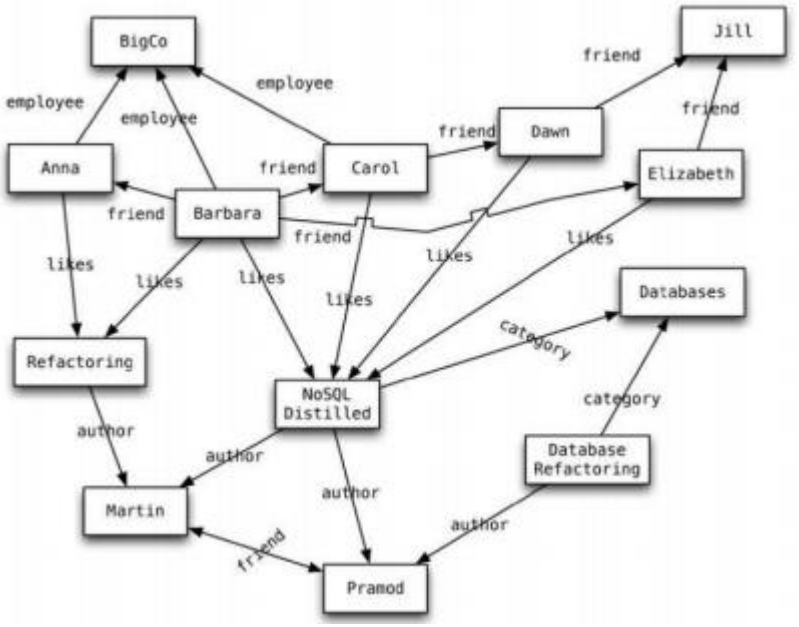



**Figura 2-11: Exemplo explicativo do modelo Orientado a Grafos**
**Fonte: (Sadalage e Fowler, 2013)**

Com base no pequeno exemplo apresentado na figura, pode-se simular uma consulta, de forma a tentar compreender como este sistema funciona. Por exemplo, caso se pretenda procurar todas as pessoas que estão empregadas pela organização *BigCo* e que gostem do livro *NoSQL Distilled* será então efetuada uma procura na base de dados devolvendo os nós correspondentes à *Barbara* e à *Carol* (Sadalage e Fowler, 2013).

O caso de aplicação apresentado é bastante simples, comparado com aqueles que têm um contexto real, como será o caso de uma loja de vendas *online* (*e-commerce*). Neste caso, poderá ser necessário, por exemplo, mostrar ao utilizador os produtos recomendados ou já comprados por outros, adotando uma base de dados desta categoria para se dar uma resposta eficaz ao utilizador (Sadalage e Fowler, 2013). A principal base de dados que utiliza esta abordagem é a Neo4J (Cattuto et al., 2013).

# 3. Métricas de Medição do Desempenho

A melhor forma de avaliar o desempenho dos SGBD que se regem pelos modelos apresentados no capítulo anterior é utilizando métricas (Davidson, 1982). As métricas definem-se como indicadores de medição da qualidade e/ou de desempenho de um sistema, aplicação ou *software*. Estas podem ser definidas qualitativamente e quantitativamente, apesar de, em termos de rigor científico, serem adotadas as métricas quantitativas, por motivos óbvios. Por norma, é adotada uma combinação de métricas que avaliam a eficácia de um processo (MSG, 2017).

Na componente experimental desta dissertação serão recolhidos valores e considerações com o objetivo de avaliar, com recurso a comparações gráficas, o desempenho de cada motor de base de dados. A avaliação de desempenho é um aspeto essencial para esta dissertação, uma vez que será através desta que serão obtidos os resultados alvo de discussão e vertidos na conclusão.

## 3.1 Perceção Humana

A primeira métrica a ser abordada nesta secção é a Perceção Humana. Esta é a métrica em que os utilizadores em geral se baseiam para formarem uma opinião e, consequentemente, avaliarem o desempenho do sistema em questão. Durante os últimos anos, foram realizados diversos estudos relativos ao comportamento humano, atendendo a três diferentes situações, com o intuito de analisar e revelar a perceção que uma pessoa obtém, quando submetida a diferentes tempos de resposta. Segundo Nielsen (Nielsen, 1994), os três limites de tempo são os seguintes:

- 0.1 segundos é o tempo limite para que um utilizador sinta que o sistema reage de forma instantânea às suas ações, descartando-se qualquer tipo de *feedback* acerca do progresso da ação, apresentando apenas o resultado obtido;

- 1 segundo é o tempo limite para que a concentração de um utilizador não seja interrompida, embora seja percetível o atraso do sistema. De forma geral, não é necessário um *feedback* especial entre o intervalo de tempo entre 0.1 e de 1.0 segundos, embora o utilizador se aperceba do atraso que a sua ação implicou sobre sistema;

- 10 segundos é o tempo limite para manter a atenção do utilizador. Para tempos de espera superiores o utilizador sente a necessidade de realizar tarefas secundárias, enquanto espera que esta termine. Assim, deve-se apresentar um *feedback* constante, revelando o progresso da ação, até que esta seja totalmente executada. Este aviso é essencial para que o utilizador tenha a noção que não ocorreu nenhum problema e, assim, não desista de esperar até que a ação seja completamente concluída.

Apesar dos estudos relativos aos tempos de perceção serem subjetivos e de terem sido desenvolvidos no domínio da usabilidade, por exemplo de uma aplicação ou sistema informático, demonstram também que os tempos de espera são interpretados de modos distintos, pelos diferentes utilizadores. Os tempos são influenciados pela ansiedade resultante do estado psicológico e emocional do utilizador em aceder, o mais rápido possível, à informação desejada (Nielsen, 1994).

Embora estes estudos abordem especificamente o caso da interação do utilizador com a *interface* gráfica de um sistema, pode-se considerar que as suas conclusões são também pertinentes no âmbito pretendido, ou seja, do tempo de resposta de uma consulta. Assim, deverá ser aplicada esta métrica na análise e discussão do desempenho, do que se propõe analisar (Nielsen, 1994).

## 3.2 Tempo de Resposta

O tempo de resposta de uma consulta é a métrica mais importante para o utilizador, que lida diariamente com uma aplicação ou sistema. Esta métrica quantitativa não é mais do que o tempo real que uma consulta ou transação demora a processar, devolver e apresentar a informação correspondente (Broadwell, 2004), (Bell, 2015).

Com base no tempo de resposta, o programador ou analista de base de dados avalia e pondera, na perspetiva do cliente, se é necessária a otimização da consulta, ou a apresentação de um *feedback* específico para a ação efetuada. Como é normal, espera-se também que, sempre que é efetuada uma consulta, os tempos de resposta sejam os mais baixos possíveis, de forma a não obrigar a que o utilizador fique à espera do seu resultado, mesmo que este seja diminuto. Tal como foi exposto anteriormente, a interpretação dos tempos de resposta depende da perceção do tempo que cada pessoa tem, assim como é influenciado pelo estado emocional no que diz respeito à necessidade de se obter os dados desejados (Nielsen, 1994).

O tempo de pesquisa depende de muitos fatores internos e externos ao motor de base de dados que se está a utilizar. Os fatores internos consistem no número de pedidos efetuados à base de dados num intervalo de tempo, na quantidade e na complexidade de informação a devolver. Por outro lado, os fatores externos podem-se resumir na velocidade, disponibilidade e capacidade de resposta do servidor onde reside a base de dados. Inclui-se também o custo e tempo de rede, caso se esteja a consultar os dados a partir da Internet, de servidores distribuídos, ligados entre si por uma rede local.

De forma a garantir que se obtém o melhor tempo de resposta possível é assim importante testar a aplicação que "consome" a base de dados, sob influência dos diferentes fatores apresentados atrás, avaliando o seu desempenho e os seus tempos de resposta obtidos, com o objetivo de apresentar uma solução que vá ao encontro dos requisitos propostos por uma organização. Com base nestes resultados poderá avaliar-se qual o fator que não corresponde ao esperado, otimizando-o.

Concluindo, será esta a métrica que se terá em especial atenção na análise de desempenho dos sistemas a desenvolver, uma vez que é a mais importante para o utilizador final e, na qual, este se focará para avaliar o desempenho de uma aplicação ou sistema, assim como por se apresentar como uma métrica quantitativa (Nielsen, 1994).

## 3.3 Armazenamento e Memória RAM

A capacidade de armazenamento e a memória *Random Access Memory* (RAM) são componentes de *hardware* fundamentais para os SGBD, uma vez que é aqui que os dados são armazenados. Apesar de esta métrica ser importante, na análise de desempenho de um sistema, não será avaliada na prova e conceito desta dissertação, devido à falta de recursos e ao elevado custo que estes componentes de *harware* apresentam.

Quanto maior for a capacidade de armazenamento de um servidor, maior será a quantidade de dados que é possível armazenar. A característica mais importante num disco de armazenamento de dados consiste na velocidade de escrita e de leitura dos dados (*input*/*output*), uma vez que esta propriedade dita a quantidade de dados que é possível processar num determinado espaço de tempo (Anikin, 2016). Pode-se ter como exemplo os discos HDD (*Hard Disk Drive*) e SSD (*Solid State Drive*), onde o primeiro geralmente contém uma taxa de desempenho bastante inferior ao segundo, conseguindo, no entanto, uma maior capacidade de armazenamento em função do valor monetário (Rizvi e Chung, 2010).

A memória RAM tem a particularidade de conseguir armazenar dados em memória *cache*, aumentando os valores de desempenho do sistema. Isto deve-se à criação de uma cópia dos dados que são mais comummente consultados, em memória *cache*, o que permite uma maior rapidez, dado que não é necessário efetuar a consulta dos dados no disco, propriamente dito. Quanto maior for a quantidade de RAM disponível num servidor, maior a quantidade de dados que se pode armazenar. Contudo, em caso de falhas de alimentação elétrica, por exemplo, podem-se perder todos os dados armazenados o que, poderá implicar um maior custo ao servidor, de forma a obterem-se novamente os dados em questão (Anikin, 2016).

Tendo em conta o acabado de expor, pode concluir-se que existem duas formas de armazenar a informação: em memória (RAM) ou em disco. A melhor forma de armazenar os dados desejados deverá ser ponderada em função do conceito do sistema e segundo os requisitos da organização. Significa isto que a informação que se pretende apresentar ao utilizador, em primeira instância, deverá ser armazenada em memória *cache*, garantindo uma maior velocidade na consulta e apresentação dos valores. Caso isto não se verifique poderá optar-se por discos SSD ou até mesmo por discos HDD. Por outro lado, se for necessário efetuar uma gestão de recursos mais controlada, pode-se utilizar uma abordagem híbrida, onde se poderá utilizar o conjunto de memórias em simultâneo.

## 3.4 Taxas de Processamento

Cada consulta implica um custo ao servidor intitulada taxa de processamento. A taxa de processamento incide diretamente sobre o CPU (*Central Processing Unit*) da máquina, onde está armazenado o motor de base de dados e os respetivos dados. Tal como na métrica, apresentada na secção anterior, esta também não será avaliada na componente experimental, uma vez que isto implicaria a alteração deste componente de *hardware* o que apresenta um elevado custo associado.

Como será fácil concluir, quanto maior for a quantidade de dados armazenados, maior será o processamento necessário para responder à consulta. Isto deve-se à necessidade de selecionar, filtrar e devolver um conjunto de valores de entre um enorme volume de dados que não é do interesse do utilizador.

Quanto maior for a taxa de processamento de um CPU melhor será o desempenho do motor de base de dados em questão, no que diz respeito ao processamento de dados. A taxa de processamento é influenciada pelo número de núcleos que um processador possui. As consultas e transações efetuadas poderão ser divididas entre os diferentes núcleos o que, consequentemente apresenta um maior desempenho do que a utilização de um único núcleo, por exemplo (Pirk et al., 2013).

Esta métrica é importante para a avaliação do desempenho de um sistema, uma vez que o processamento de dados, para posterior armazenamento e consulta, é uma das etapas fundamentais, quando se pretende aceder aos dados necessários. No decorrer desta dissertação, todos os componentes *hardware* utilizados na elaboração da componente experimental serão apresentados, conforme a sua pertinência.

## 3.5 Volume de Dados Utilizados

O volume de dados presente numa base de dados representa uma métrica que dita, de forma geral, todas as demais documentadas nas secções anteriores. Tal como se teve a oportunidade de observar, foi o crescente volume de dados a armazenar que determinou a necessidade de evoluir, resultando no desenvolvimento e apresentação de novos modelos de dados e, consequentemente, de novos sistemas de gestão de bases de dados.

Como se pode deduzir, o tempo de execução e a taxa de processamento de uma consulta sobre um conjunto de centenas de registos é totalmente diferente do que efetuar uma consulta sobre milhões de registos. Desta forma, serão elaborados diversos testes com diferentes quantidades de dados para se analisar o comportamento de ambos os sistemas a testar.

Isto será efetuado com o objetivo de se analisar a discrepância dos tempos de resposta e da taxa de processamento de uma consulta, em função do volume de dados. Os dados que serão utilizados para se efetuar os testes desejados terão como objetivo corresponder ao mesmo tipo de dados que se pretende armazenar, atendendo aos diferentes tipos problemas a ter em consideração.

# 4. Descrição do Problema

Este capítulo tem como objetivo apresentar o problema prático a ser abordado nesta dissertação. Com isto, serão apresentadas as diferentes vertentes do sistema poliglota, tendo como referência o sistema único. As divisões serão refletidas através de um modelo conceptual de dados, assim como através de uma possível arquitetura da aplicação.

## 4.1 Problema

O problema a abordar nesta prova de conceito terá como base um sistema complexo, equiparado a uma aplicação ou sistema real. A abordagem ideal seria efetuar a comparação do desempenho de um sistema em funcionamento, que já não corresponde aos requisitos inicialmente impostos, com o sistema poliglota a construir em busca de uma alternativa viável.

Apesar desta perspetiva ser a mais indicada, é bastante difícil comparar sistemas complexos em atual funcionamento em organizações de média ou grande envergadura. Tal, deve-se à não permissão da partilha do sistema e dos respetivos dados pela grande maioria das organizações. Devido a este inconveniente, o problema a abordar nesta dissertação, será, portanto, fictício e terá como base o espírito crítico e o conhecimento dos conceitos de bases de dados, tentando assemelhar-se, o máximo possível, a um problema atual e real.

O sistema, apesar de fictício, deverá apresentar um nível de complexidade capaz de estabelecer os requisitos mínimos que uma aplicação impõe. Os requisitos podem-se caracterizar como a consistência, disponibilidade e a grande capacidade de armazenamento de dados.

Pretende-se, portanto, simular a base de dados que alberga um sistema semelhante ao de uma transportadora de encomendas. O envio de correspondência e encomendas através de transportadoras tem crescido ao longo do tempo, devido à simplicidade e rapidez do envio. O problema prático a apresentar tem por base o caso concreto da transportadora alemã DHL, com sede na cidade de Bona que, ano após ano, regista um crescimento sustentável.

A Figura 4-1, retirada do relatório anual da organização (Deutsche Post DHL Group, 2017), permite dar a conhecer as operações da empresa e o respetivo volume de transações, ao nível global. Com base na figura, é possível saber que foram processadas, em 2016, cerca de quatro milhões de encomendas diárias, através de milhares de pontos estratégicos, dispersos por todo o mundo. Estes locais são representados como pontos de empacotamento, venda, centros de distribuição e de entrega ao cliente. A quantidade de encomendas é de tal forma elevada que é necessário desenvolver um sistema capaz de armazenar um tal volume de dados, correspondentes às transações havidas.

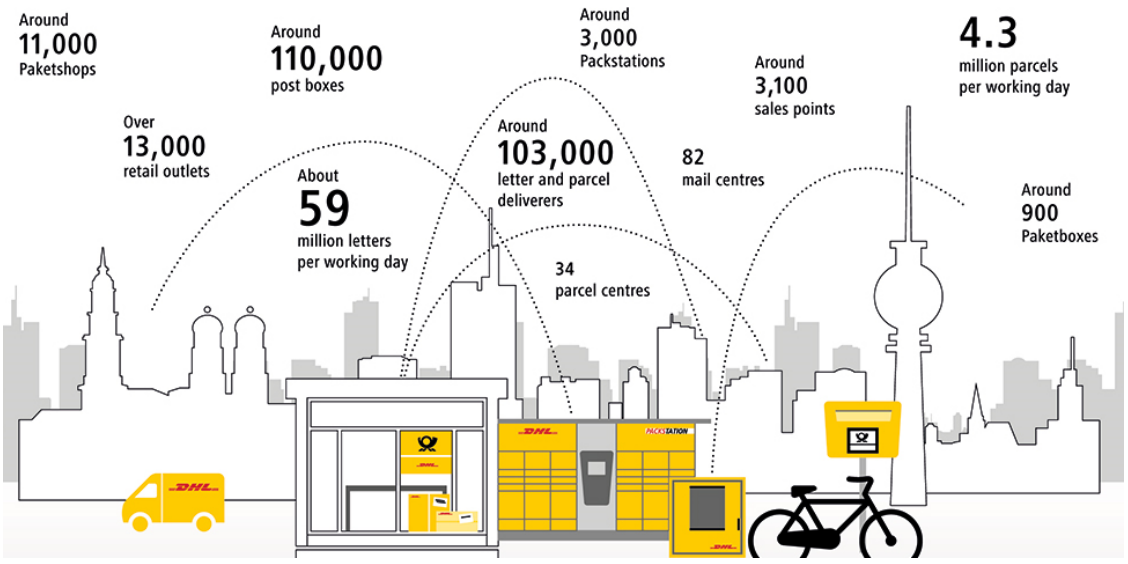



**Figura 4-1: Volume de negócio da organização DHL**
**Adaptado: 2016 Anual Report, DHL**

Este cenário tem como objetivo levar ao ponto fulcral desta dissertação, ou seja, à implementação de um sistema único *versus* um sistema poliglota que permita adotar uma nova solução ao nível de base de dados, representado na Figura 4-2. Pretende-se assim, apresentar e

adotar para cada vertente aplicacional, Informação Encomendas, Faturação, Logs e Coordenadas GPS (*Global Positioning System*), um motor de base de dados correspondente ao tipo de dados a armazenar compondo um sistema poliglota. Isto tem como intuito verificar se existe efetivamente um maior desempenho destes sistemas, face aos que apenas utilizam um único tipo de base de dados para guardar todos os dados, independentemente do seu tipo e origem.

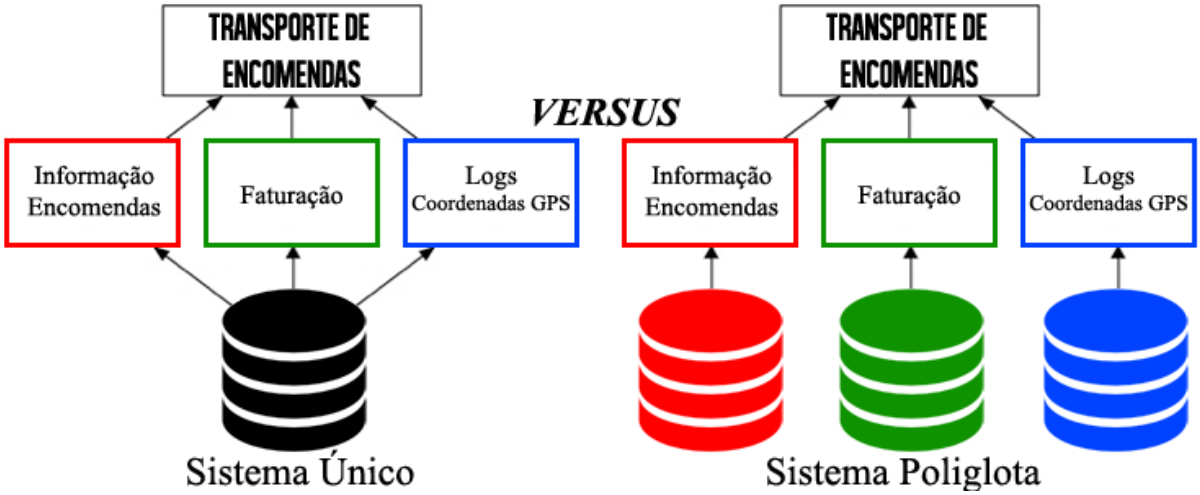



**Figura 4-2: Sistema único *versus* sistema poliglota**

Com base na Figura 4-2, a vertente aplicacional identificada a vermelho, deverá possibilitar o armazenamento dos dados relativos a uma encomenda, assim como o registo da localização, conforme os pontos estratégicos atingidos. Através da disponibilização de um código de procura da encomenda, o cliente poderá consultar e visualizar onde esta se encontra. Isto será também útil, dada a possível demora na entrega da encomenda, uma vez que muitas têm como origem o outro lado do Planeta, podendo decorrer alguns dias até chegar ao seu destino. Posteriormente, com esta informação, pretende-se também providenciar uma extensão que permita a integração em *websites* de vendas *online*. Desta forma, será apresentada a situação em que se encontra a encomenda, sem ser necessário navegar até ao *website* oficial da organização que efetua a entrega das respetivas encomendas.

Numa segunda vertente, a verde, tenciona-se armazenar todos os dados relativos à faturação das encomendas. A faturação terá por base a origem, destino, assim como taxas alfandegárias adotadas pelos diferentes países e pelos diferentes produtos transportados, dependendo das características que uma encomenda apresenta, tais como o peso, largura, altura e comprimento.

Por último, na vertente aplicacional representada a azul, deseja-se também obter uma solução fiável que permita a inserção em massa de *logs* de procura e acesso às diversas encomendas, por parte dos clientes. Engloba-se nesta componente final o armazenamento de coordenadas obtidas por um dispositivo GPS instalado nas carrinhas de entrega, assim que esta se encontre num raio de dez quilómetros da morada do cliente. Estas coordenadas têm como objetivo prover uma aplicação móvel, através da qual o cliente poderá acompanhar a chegada da sua encomenda. Desta forma, o cliente será alertado para a localização do funcionário e garantir a receção da sua encomenda, tornando-se uma mais-valia. Pretende-se também analisar a informação de forma a aumentar a produtividade nas entregas e a experiência do cliente quanto ao serviço prestado, assim como para testar a qualidade do serviço.

## 4.2 Modelo Conceptual de Dados

O modelo conceptual de dados tem como objetivo apresentar a base de dados a desenvolver, com as respetivas cardinalidades e relacionamentos entre as entidades, de forma a que se possa compreender numa primeira abordagem a estruturação da aplicação e, posteriormente, o modelo físico de dados, que será apresentado no capítulo seguinte.

Tendo por base a descrição do problema efetuada na secção anterior, é possível apresentar um esboço do sistema único, assim como a divisão do sistema poliglota a desenvolver, através do diagrama representado pela Figura 4-3.

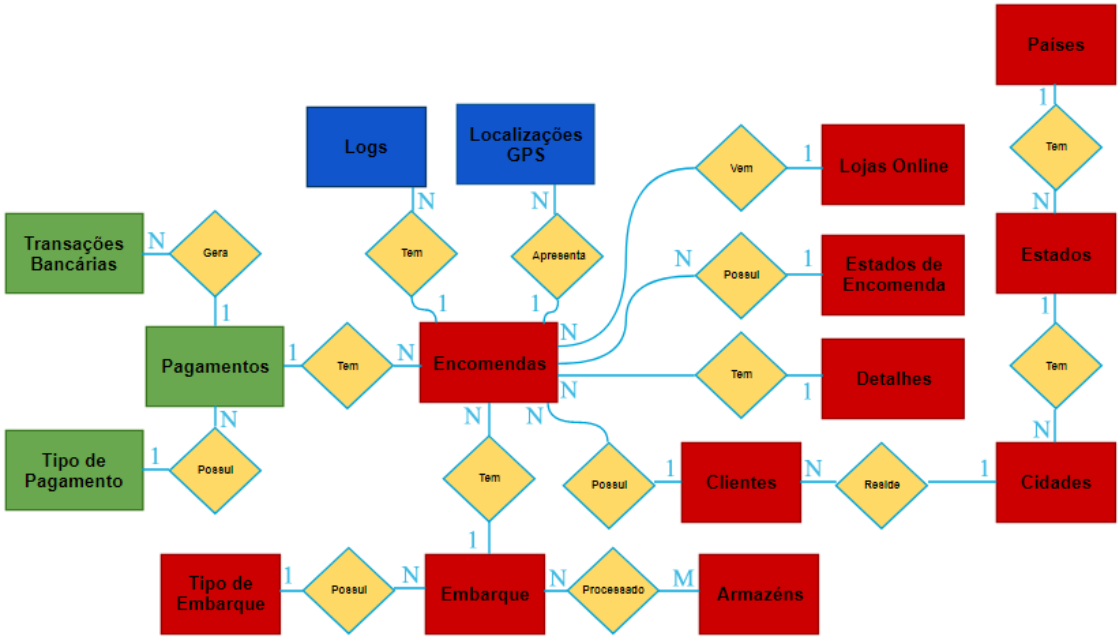



**Figura 4-3: Modelo Conceptual de Dados**

O diagrama reflete, desde já, a divisão da aplicação, conforme apresentado anteriormente, através das cores verde, vermelha e azul. Portanto, é possível visualizar que o sistema único é composto por todas as entidades presentes na figura ou seja: Encomendas, Embarque, Tipo de Embarque, Armazéns, Clientes, Cidades, Estados, Países, Lojas Online, Estados de Encomenda, Detalhes, Logs, Localizações GPS, Pagamentos, Tipo de Pagamento e Transações Bancárias.

Por outro lado, o sistema poliglota é divido em três partes. A primeira é referente às Informações da Encomenda, sendo refletida pelas entidades a vermelho: Encomendas, Embarque, Tipo de Embarque, Armazéns, Clientes, Cidades, Estados, Países, Lojas Online, Estados de Encomenda e Detalhes. A segunda parte, representada a verde, refere-se às entidades: Pagamentos, Tipo de Pagamento e Transações Bancárias. A terceira e última parte é referente aos Logs e Coordenadas GPS e é refletida pelas entidades a azul.

Concluindo, como já referido anteriormente, por cada divisão presente no sistema poliglota será adotado um SGBD. Assim, a representação dos relacionamentos entre entidades, visível na Figura 4-3, é meramente informativa e representativa do modelo relacional único. Quanto a cada divisão, consoante os SGBD a utilizar, serão adotadas diferentes formas de relacionar/estruturar os dados conforme as melhores práticas de cada um.

## 4.3 Sugestão de Arquitetura da Aplicação

Após a apresentação do problema é possível perceber através da Figura 4-2 e da Figura 4-3, que poderá ser pertinente consultar simultaneamente os diversos motores de base de dados, que compõem o sistema poliglota, de modo a obter-se a informação global presente nos mesmos. Uma vez que se vão adotar diversos tipos de SGBD e, consequentemente, diferentes características e sintaxes de consulta, não será possível efetuar os comandos, de forma simultânea, na camada de base de dados.

Deste modo, propõe-se uma possível arquitetura dividida em camadas lógicas, semelhante a um sistema MVC (*Model*, *View*, *Controller*), para o problema apresentado (Selfa, Carrillo e Boone, 2006), (Deacon, 2009). Apesar desta arquitetura não ser implementada no decurso da componente experimental, visto que não vai ao encontro aos objetivos propostos, poderá ser uma hipótese pertinente a adotar em sistemas poliglotas pelas razões a apresentar.

Esta arquitetura permite, de forma resumida, a divisão de um sistema em três camadas diferentes que se encarregam de separar a *interface*, da lógica e do acesso aos dados. Deste modo, é facilitada a gestão e evolução dos sistemas dada a independência destas camadas (Selfa, Carrillo e Boone, 2006). As camadas lógicas que compõem esta possível arquitetura são mostradas na Figura 4-4.


**Figura 4-4: Possível arquitetura do sistema poliglota**

Como se pode observar na figura, o sistema proposto é composto por três motores de base de dados, atendendo ao sistema poliglota representado pela Figura 4-2. Estes interagem diretamente com a camada de acesso aos dados, que poderá ser implementada através de uma *framework* e com os respetivos *drivers*, semelhante à camada *controller*. Os *drivers* têm como objetivo permitir a conexão às bases de dados respetivas, permitindo a leitura e escrita de dados para a BD correspondente. Posteriormente, tem-se a camada de negócio, que poderá ser encarada analogamente à camada *model*. Nesta camada é aplicada toda a lógica de gestão e controlo do negócio. Por último, a camada de interface, ou seja, a *view*, onde os utilizadores finais podem visualizar a informação disponibilizada.

As ligações presentes na arquitetura da Figura 4-4, representam pedidos lógicos que se efetuam entre as camadas MVC, de forma a dar resposta aos pedidos de leitura e escrita. A divisão de um sistema por camadas lógicas mostra uma maior simplicidade na estruturação e organização dos componentes lógicos que compõem a aplicação.

# 5. Sistemas de Gestão de Base de Dados Utilizados

Este capítulo tem como objetivo sintetizar todos os SGBD em uso na prova de conceito. Numa fase inicial serão apresentados os diversos critérios de seleção dos motores de base de dados e então, posteriormente, a seleção das tecnologias utilizadas.

Para cada problema que se propõe tratar, constituindo, no seu conjunto, a prova de conceito, será apresentada uma comparação detalhada das diversas alternativas em termos de sistemas de gestão de base de dados, optando-se pelo mais adequado ao problema em questão.

## 5.1 Critérios de Seleção

A escolha dos motores de base de dados, a adotar na componente experimental, será efetuada atendendo a alguns critérios de seleção. Pretende-se com isto apresentar uma solução, ao nível das bases de dados, que melhor se adapte e satisfaça os requisitos dos problemas a tratar, tendo por base o menor custo possível.

Num âmbito geral, os critérios de seleção baseiam-se no tempo a despender na pesquisa e análise do *software* e *hardware* a utilizar, no investimento financeiro e nos recursos humanos que a implementação do sistema a desenvolver implica.

Para o sistema poliglota, o principal critério recai na gratuitidade dos motores de base de dados, das tecnologias e dos serviços a utilizar. Segue-se, como prioridade, a seleção dos SGBD de

modo a corresponder ao tipo de dados a armazenar, assim como que apresente características, princípios e funcionalidades que permitam atingir os requisitos desejados. Também se irá ter em consideração a simplicidade de instalação, modelação, consulta, configuração e utilização do SGBD. Por último, serão consideradas as informações pertinentes, disponíveis nos meios oficiais de cada sistema e provenientes da revisão bibliográfica, fundamentando todas as escolhas tomadas.

Em resumo, espera-se que seja possível desenvolver um sistema sem que este implique um grande investimento, embora ofereça todas funcionalidades e princípios pretendidos.

## 5.2 Microsoft SQL Server 2014

O Microsoft SQL Server 2014 foi o motor de dados relacional escolhido para o sistema único proposto. Tal como referido na descrição do problema, este SGBDR terá como objetivo armazenar e gerir todos os dados relativos às encomendas, faturas, coordenadas GPS e *logs*, relativas à aplicação de entrega de encomendas descrita.

Assim, este sistema tem como requisitos o armazenamento e gestão de diversos tipos e de volumes de dados. Visto que se está perante uma única base de dados, tem-se como requisito a aplicação de princípios transacionais que permitem manter a consistência, coerência e integridade dos dados. Por último, assume-se como requisito a utilização de um SGBD empresarial, pago e com suporte por parte de quem o disponibiliza, de forma a replicar um sistema de uma organização com grandes proporções.

Tendo em conta os requisitos apresentados, optou-se por descartar os sistemas NoSQL uma vez que um sistema desta proporção apresenta relacionamentos entre entidades, o que se traduz num baixo desempenho por parte destes sistemas. Desta forma, tomou-se como alternativas os SGBDR, empresariais e pagos, Microsoft SQL Server 2014 e o Oracle. Neste caso em particular, optou-se por utilizar o primeiro motor de base de dados *versus* o Oracle, devido a um maior contacto por parte do autor com esse SGBD, assim como na limitação dos recursos para a aquisição de um destes motores de base de dados. Com isto, podem-se especificar as diversas características que esta ferramenta apresenta, assim como justificar e enquadrar a utilização deste SGBD no problema proposto.

Além do principal motivo apresentado, a escolha deste sistema tem por base a quantidade de organizações que a utilizam, pela particularidade de ser um sistema de licença paga com suporte por parte da Microsoft, por ser um motor de base de dados relacional e uma ferramenta amplamente utilizada e conhecida (iDatalabs, 2017).

O Microsoft SQL Server 2014 baseia-se nos princípios do modelo relacional o que permite efetuar o relacionamento entre entidades, garantindo a integridade dos dados. Este SGBDR aplica os princípios ACID permitindo a consistência e coerência dos dados quando efetuadas operações de escrita. As operações, efetuadas na base de dados em questão, serão efetuadas através da linguagem de consulta SQL, característica dos modelos relacionais, através dos comandos DCL, DDL, DML já apresentados.

A um nível funcional, o SGBDR oferece vários níveis de segurança, tendo por base diferentes tipos de acessos, com diversos tipos de permissões que poderão dar ou não acesso a determinadas funcionalidades. Do mesmo modo, efetua a encriptação das bases de dados e dos respetivos *backups* sendo estes periódicos (Nielsen, White e Parui, 2009). Quanto à sua arquitetura, apresenta um sistema de otimização de consultas, assim como serviços, extensões e funcionalidades que permitem a gestão e a análise do desempenho do sistema.

Outra característica deste motor de base de dados traduz-se na capacidade de se armazenarem dados na ordem dos *terabytes*, embora acima deste valor possam verificar-se dificuldades quanto ao desempenho desejado (Nielsen, White e Parui, 2009). Como não se pode deixar de referir, uma correta utilização e configuração do motor de base de dados, ao nível do *software* e *hardware* e a sua manutenção, são uma mais-valia no que diz respeito ao funcionamento da base de dados. Para isto, pode-se consultar a extensiva documentação que a ferramenta apresenta.

Por fim, este motor de base de dados relacional tenta evoluir consoante as necessidades, apresentando nas diversas versões melhorias evidentes para as organizações que utilizam este SGBDR para a gestão dos dados das suas aplicações (Fojtik, Podesva e Gebauer, 2014).

## 5.3 PostGreSQL

O PostGreSQL, ou simplesmente Postgres, será o primeiro SGBD a ser apresentado, que irá constar no sistema poliglota. Esta ferramenta é incluída nesta abordagem com o objetivo de armazenar toda a informação relativa à faturação das encomendas, englobando os valores, métodos e todo o mecanismo transacional de pagamento.

Neste caso em particular, tem-se como requisito o armazenamento de dados, constituídos maioritariamente por dados estruturados, que têm como origem as faturas e movimentos bancários o que exige suporte dos sistemas OLTP (*Online Transaction Processing*) (Kaur e Rani, 2015), (Conrad, 2006). Daqui, deduz-se que se tratam de dados sensíveis, confidenciais e que requerem uma preocupação adicional quanto à coerência, consistência, integridade, normalização e operações transacionais dos dados.

Assim, optou-se por colocar de parte os sistemas NoSQL pois, perante as características dos dados, verificam-se como mais adequados os sistemas relacionais uma vez que os dados a armazenar são estruturados e requerem um maior controlo transacional (Chickerur, Goudar e Kinnerkar, 2015), (Jiménez-Peris et al., 2016), (Srivastava e Shekokar, 2016). De acordo com os critérios de seleção apresentados, descartaram-se os sistemas relacionais pagos, tomando-se como alternativa os sistemas relacionais grátis e de código aberto (*open-source*) tais como o PostGreSQL, MySQL e FireBird.

Efetuando a comparação do SGBDR PostGreSQL com as alternativas referidas, foi tomada a decisão de se utilizar o primeiro devido a ser um sistema objeto-relacional, o que não é o caso das restantes. Esta propriedade permite a construção de novos tipos de dados, assim como relacionamentos hierárquicos entre tabelas, o que se apresenta como uma mais valia no armazenamento de dados, mesmo que numa primeira abordagem não se utilize esta propriedade (Smith, 2015).

Desta forma, optou-se pelo PostGreSQL para se armazenar os dados relativos à vertente da faturação das encomendas. Este, tal como já referido, é classificado como um SGBD objeto-relacional de código aberto (*open-source*), desenvolvido no início de julho de 1996, tendo sido lançadas atualizações periódicas que dão resposta às novas necessidades (Veen, Waaij e Meijer, 2012), (Worsley e Drake, 2002), (Conrad, 2006), (PostGreSQL, 2017a).

Este sistema de gestão de base de dados relacional opera sobre diversos tipos de sistemas operativos tais como o Linux, UNIX e Windows e apresenta os mais diversos tipos de dados (Worsley e Drake, 2002). Além disto, é uma ferramenta que disponibiliza a replicação assíncrona, recuperação da base de dados tendo por base gravações periódicas e automatizadas, pontos automáticos de *backup*, um sofisticado optimizador de consultas e, por último, um mecanismo que possibilita que o sistema seja tolerante a falhas (PostGreSQL, 2017a), (Conrad, 2006), (Smith, 2015), (Tezer, 2014).

Após a descrição das características gerais do SGBD, acrescenta-se que este gere toda a concorrência de dados e de utilizadores através de um sistema denominado *Multiversion Concurrency Control* (MVCC). O controlo de concorrência multiversão permite que por cada transação de escrita efetuada se gere uma cópia do registo afetado. Deste modo, é criado um limite de abstração perante as transações de leitura, uma vez que estas serão consultadas na cópia gerada, até a transação ser completamente efetuada. Outra grande vantagem que o PostGreSQL apresenta é o suporte de extensibilidade, ou seja, qualquer programador poderá desenvolver uma extensão e integrá-la de forma a criar e facilitar certas funcionalidade como, por exemplo, o desenvolvimento de procedimentos armazenados (Smith, 2015), (Tezer, 2014).

Por fim, pode-se realçar que na eventualidade de ser necessário migrar a base de dados para um outro SGBDR, o PostGreSQL permite, através de uma funcionalidade própria, a sua fácil exportação e posterior integração (PostgreSQL, 2017b).

Concluindo, pode-se afirmar que, segundo a análise efetuada, o motor de base de dados PostGreSQL se adequa ao tipo de problema a resolver, ou seja, o armazenamento de dados sensíveis, como os dados relativos à faturação das encomendas.

## 5.4 MongoDB

O SGBD NoSQL MongoDB foi o escolhido para armazenar os dados relativos às encomendas no sistema poliglota proposto. Como é evidenciado na Figura 4-3, o MongoDB (a vermelho) apresenta-se como o núcleo da aplicação, uma vez que é aquele que faz a ligação entre os dois outros sistemas que compõem o sistema poliglota. Assim, caso seja necessário obter a informação presente nos diversos SGBD, este fará de intermediário na "ligação" entre os dados dos diversos motores de base de dados.

Relembrando o problema em questão, tomam-se como requisitos o armazenamento de grandes volumes e tipos de dados ao longo do tempo, um grande desempenho na leitura/escrita, alta disponibilidade dos dados, a ausência de um esquema rígido (*schema-free*) e a capacidade de suportar uma base de dados distribuída. É importante referir que se pretende obter uma solução que permita a adição e remoção de atributos, conforme o desejado, sem causar transtornos para os dados já presentes na base de dados. Tendo por base a descrição do problema, efetuada no capítulo anterior e, relembrando, pretende-se providenciar dados a uma extensão a integrar em *websites* de vendas *online*, sendo de valorizar um SGBD que permita a fácil integração e disponibilização de dados para estes tipos de sistemas.

Dado o tipo de dados e as circunstâncias específicas do problema proposto, foram descartados os diversos sistemas de gestão de base de dados relacionais, uma vez que não satisfazem os requisitos necessários (ausência de esquema rígido, alta disponibilidade, armazenamento de diversos tipos de dados, etc.). Pode-se também afirmar que os dados a armazenar, neste caso em específico, são pouco sensíveis visto que se tratam de dados maioritariamente informativos. Deste modo, optou-se por escolher um SGBD com um menor controlo dos dados, no que diz respeito à integridade e consistência dos mesmos (Zhao et al., 2013), (Nyati, Pawar e Ingle, 2013).

Dado estas considerações, analisaram-se diversas alternativas baseadas no modelo NoSQL. Numa primeira abordagem, colocaram-se de parte os SGBD NoSQL que não se adequavam ao tipo de dados e ao contexto do problema como, por exemplo, os motores de base de dados orientados a grafos (e.g. OrientDB e Neo4j) e os motores de base de dados de família de colunas (e.g. Cassandra e HBase). Foram ainda analisadas e ponderadas outras alternativas, como o MongoDB, Redis e, por fim, o CouchDB.

O SGBD Redis foi excluído logo à partida, uma vez que tem como principal objetivo armazenar e gerir dados em *cache*. Este tipo de abordagens adapta-se, principalmente, a sistemas onde é obrigatória a pouca latência no momento da consulta dos dados, não sendo este o caso (Redis, 2017), (Chen et al., 2016). No entanto, poderá ser uma alternativa bastante interessante para sistemas onde é necessário a consulta e apresentação dos dados em tempo real.

Posteriormente, analisou-se a opção CouchDB que inicialmente revelou capacidades e características bastante interessantes. Após alguma análise, revelou diversas debilidades na forma como lida com as falhas no sistema e, consequentemente, na consistência dos dados. Por

último, apresenta também uma desvantagem relativamente ao *sharding* (técnica/método utilizado para escalar horizontalmente os dados, por diversos servidores, de forma a aliviar a carga de um servidor) visto que não possui um mecanismo próprio (CouchDB, 2017), (Prasad e Gohil, 2014), (Datt, 2016), (Saino, Psaras e Pavlou, 2016).

Apresentados os motivos pelos quais não foram utilizadas as alternativas Redis e CouchDB, será agora momento de apresentar e caracterizar o SGBD escolhido. O MongoDB é um sistema grátis e de código aberto, que se classifica como uma base de dados orientada a documentos, implementada em C++. Este sistema não-relacional começou a ser desenvolvido em 2007, inicialmente como um sistema *Platform as a Service* (PaaS), sendo posteriormente apresentado como um motor de base de dados que armazena os dados em coleções de documentos BSON.

Numa primeira perspetiva sobre este SGBD, pode-se referir que este opera sobre diversos sistemas operativos, assim como disponibiliza serviços, extensões e *drivers* de acesso e modelação de dados. O motor de base de dados escolhido tem a particularidade de armazenar toda a informação através de ficheiros, que adotam a estrutura JSON, o que permite que haja uma maior rapidez na consulta dos dados em relação aos sistemas relacionais (Ming Wu, 2015), (Kanoje, Powar e Mukhopadhyay, 2015). Além disto, é também de salientar que apresenta uma estrutura de dados livre de esquemas, (*schema-free*) permitindo a sua alteração, ou seja, a adição/remoção de campos, sem qualquer transtorno, o que se torna numa mais-valia para o problema em questão.

O MongoDB tem também a capacidade de conseguir efetuar uma eficiente escalabilidade horizontal, o que poderá permitir a divisão dos dados em servidores, por país de entrega, por exemplo. Desta forma, é possível configurar as máquinas, conforme as necessidades, no que diz respeito ao *hardware* a adotar. Outra característica importante deste motor de base de dados é a capacidade de efetuar a replicação dos dados em diversos servidores de *backup*. Assim, é disponibilizada uma cópia de segurança onde se pode obter a informação pretendida sempre que necessário, garantindo a alta disponibilidade dos dados, mesmo que o servidor principal esteja indisponível (MongoDB, 2017).

Por último, a preferência na escolha do MongoDB deveu-se à capacidade de estruturação da informação, uma vez que apresenta os dados numa estrutura JSON. Com isto, é possível prover diretamente micro serviços, extensões, APIs (*Application Programming Interface*) e

*frameworks* sem qualquer perda de tempo na conversão e/ou construção da estrutura de dados a enviar.

Concluindo, pode-se afirmar que, neste caso em específico e segundo a análise efetuada, o motor de base de dados MongoDB apresenta as melhores características para armazenar os dados relativos à vertente representada a vermelho na Figura 4-2.

## 5.5 Cassandra

O último SGBD a fazer parte do sistema poliglota é o Cassandra. Este motor de base de dados irá permitir armazenar e gerir os dados relativos às coordenadas GPS, assim como dos *logs* das consultas, por parte do utilizador, relativos ao estado de trânsito de uma encomenda.

Este problema, em particular, apresenta como requisito o armazenamento de grandes volumes de dados, de forma constante, provenientes dos casos falados. Uma vez que esta vertente da aplicação é sobretudo direcionada ao cliente, espera-se um grande desempenho na leitura dos dados, assim como a alta disponibilidade.

Dados os requisitos apresentados atrás, foram excluídos de imediato os sistemas de gestão de bases relacionais. A razão pela qual se tomou esta decisão teve como fundamento as debilidades apresentadas no desempenho dos SGBDR no armazenamento e gestão de grandes volumes de dados. Adiciona-se também que os dados a armazenar não se apresentam como sensíveis nem confidenciais, pelo que poderão ser armazenados sob princípios menos rígidos pertencentes ao NoSQL. Foram assim colocadas de parte diversos SGBD tais como o MySQL, MariaDB, FireBird e até a possibilidade de integração deste problema no PostGreSQL.

Com isto, optou-se por se adotar uma ferramenta que se rege pelos princípios do NoSQL. Assim, foi equacionada a utilização de um dos seguintes SGBD: Cassandra, HBase, MongoDB, OrientDB e o Redis. Com isto, foram analisados os diversos sistemas, representando assim um resumo do desempenho das mesmas através da Figura 5-1.

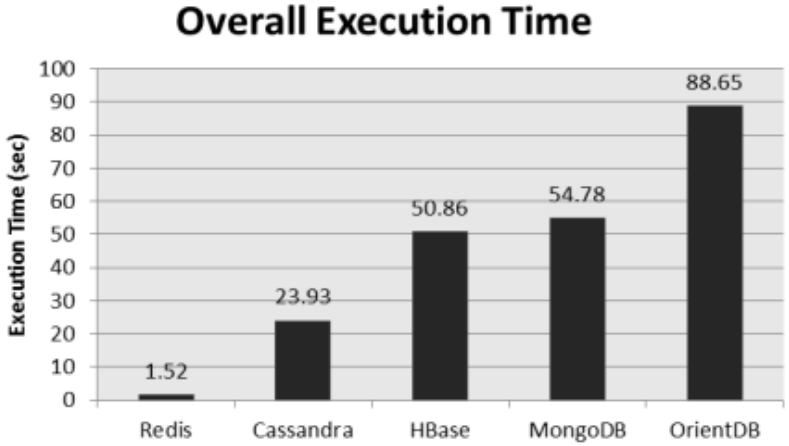



**Figura 5-1: Análise sumária da execução de operações de leitura e escrita**
**Fonte: (Abramova, Bernardino e Furtado, 2014)**

Tal como no sistema apresentado na secção anterior, o SGBD Redis apresenta-se como o motor de base de dados com melhor tempo de execução. No entanto, foi descartado devido à forma como armazena os dados, ou seja, em memória RAM. Dado esta característica indesejável, o Cassandra apresenta-se como o SGBD que apresenta maior desempenho perante os restantes (Abramova, Bernardino e Furtado, 2014). Esta seleção é também consolidada através da análise de outras avaliações de desempenho, estas com uma maior abrangência, com incidência nos processos de leitura, escrita, carga e latência sobre as consultas de dados, validam-se e fortalecem-se os resultados apresentados na Figura 5-1 (Point, 2016), (Rabl et al., 2012), (Hewitt, 2011), (Ahamed, 2016).

O sistema de gestão de base de dados Cassandra foi desenvolvido pelo Facebook, de forma a apresentar melhores resultados no desempenho das suas consultas, mais concretamente na procura e apresentação de dados baseado num campo de pesquisa (Lakshman e Malik, 2009), (Lakshman e Malik, 2010) (Gopal e Bharat, 2016). Este, pode-se classificar como um SGBD não relacional, orientada a colunas, de código aberto e que apresenta uma linguagem de consulta declarativa própria denominada *Cassandra Query Language* (CQL).

O motor de base de dados escolhido apresenta características de uma base de dados distribuída com alta disponibilidade. É altamente tolerante a falhas devido à sua arquitetura *peer-to-peer*, replicando a informação armazenada pelos diversos nós funcionais e de *backup*. Apresenta também, ao nível funcional, uma grande eficiência de escrita e de leitura, capaz de escalar até milhões de transações por segundo, indo de encontro ao pretendido para o problema em questão (Murazza e Nurwidyantoro, 2016), (Chebotko, Kashlev e Lu, 2015), (Sendir et al., 2016), (Hewitt, 2011).

Concluindo, a ferramenta Cassandra foi o motor de base de dados escolhido para armazenar toda a informação relativa às coordenadas de GPS e aos *logs* de pesquisa sobre as encomendas por parte do cliente final, fechando o sistema poliglota.

## 5.6 Síntese

Após a escolha dos SGBD a utilizar na componente experimental, pode-se sintetizar o sistema poliglota através da Figura 5-2. Esta figura tem como objetivo apresentar os SGBD escolhidos para cada vertente de modo a armazenar os dados correspondentes, para cada sistema proposto.

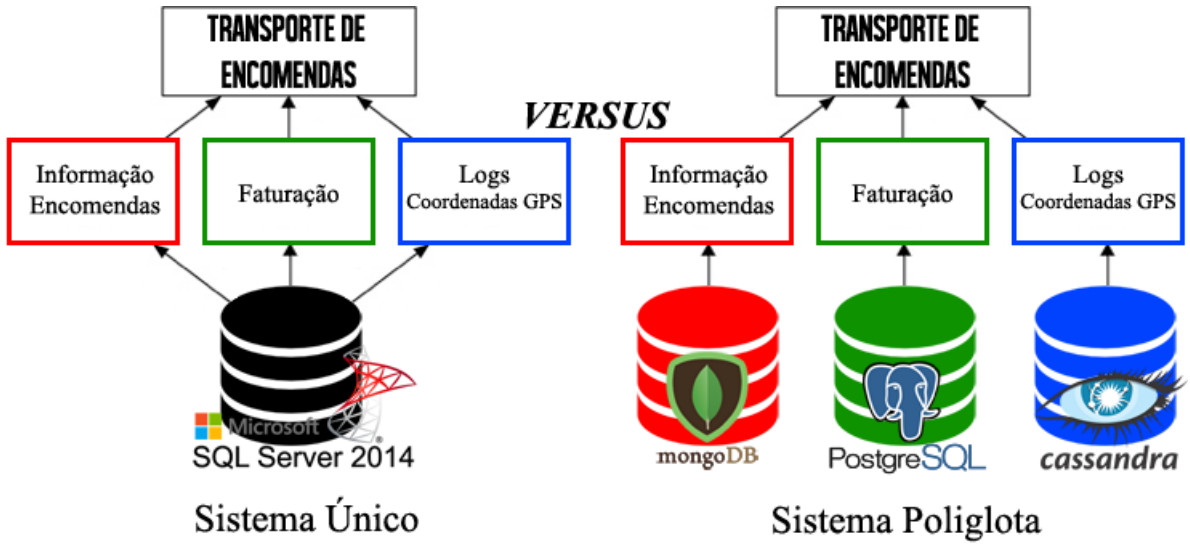



**Figura 5-2: SGBD adotados nos diferentes sistemas (único e poliglota)**

Quanto à implementação do sistema único, tal como se observa na figura, será utilizado o SGBDR Microsoft SQL Server 2014, que tem como objetivo o armazenamento de todos os dados gerados na aplicação.

Sob outra perspetiva, à direita, observa-se o sistema poliglota, dividido em três vertentes, solucionadas com recurso a uma base de dados por cada uma. A vertente relacionada com a informação de encomendas, a vermelho, será implementada através da utilização do SGBD MongoDB. A verde, na faturação será utilizado o PostGreSQL. Por último, a azul, será utilizado o Cassandra para se armazenarem os dados relativos às coordenadas de GPS e Logs.

Assim, estão reunidas todas as condições para se elaborar a componente experimental desta dissertação.

# 6. Prova de Conceito

No presente capítulo serão apresentadas todas as fases da implementação da componente experimental desta dissertação. Numa primeira abordagem, serão tecidas algumas considerações iniciais, importantes para a caracterização e configuração do sistema construído. Posteriormente, nas fases seguintes, será documentado o processo de implementação do problema proposto, a geração de dados, as consultas a efetuar para testar ambos os sistemas e, por fim, será realizada a apresentação e análise dos resultados obtidos.

## 6.1 Considerações Iniciais

A prova de conceito a desenvolver será dividida em duas componentes. Esta divisão tem como objetivo a análise comparativa dos dois sistemas que se regem por abordagens diferentes. O foco principal, a ter em conta neste trabalho, será a recolha e análise dos tempos de consulta obtidos nos diferentes SGBD, através de diferentes consultas, aplicadas a diferentes volumes de dados.

Pretende-se, portanto, efetuar consultas individuais e globais de modo a recolher e analisar os tempos de consulta obtidos. Após a divisão do problema, retratada na Figura 4-3, de uma base de dados única num sistema poliglota, serão efetuadas consultas individuais a cada motor de base de dados que compõem este sistema. Posteriormente, seguem-se as consultas de âmbito global, onde se pretende consultar e apresentar, de forma instantânea e sequencial, a informação presente no conjunto de bases de dados que compõem o sistema. Em paralelo, serão efetuadas

consultas no sistema relacional único, composto por todas as entidades, de modo a comparar os tempos de consulta de ambos os sistemas, relativos aos diferentes volumes de dados.

Estes resultados serão fundamentais para se apurar se a adoção de um sistema poliglota poderá ser uma mais-valia em sistemas complexos. Resumidamente, será efetuada uma comparação de desempenho das duas componentes desenvolvidas, quanto ao tempo de resposta das consultas propostas.

Assim, conforme descrito no capítulo quarto, a primeira componente será composta por um único motor de base de dados, o Microsoft SQL Server 2014, que irá armazenar todos os dados provenientes de um sistema que gere a entrega de encomendas. A segunda componente experimental consiste na implementação de um sistema poliglota composto pelos motores de base de dados PostGreSQL, MongoDB e Cassandra. As configurações de *software* e *hardware* adotadas para suportar ambos os sistemas são as seguintes:

| | |
|---:|---|
| **Sistema Operativo** | Windows 10 – 64bits |
| **Processador** | Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz |
| **Número de Núcleos** | 4 |
| **Memória RAM** | DDR3 – 1600 6 GB |
| **Disco** | HDD – 500GB |

Apesar de não serem estas as especificações ideais (longe disso), para um servidor de base de dados de alto desempenho, é esperada a obtenção de dados credíveis e suscetíveis de análise. Tal como referido, ambas as componentes serão testadas de forma equivalente, tendo em consideração os mesmos dados. Pretende-se, com esta abordagem, analisar os sistemas, avaliando a correlação entre volume de dados e tempo de consulta.

Por último, será importante referir que não se replicará um sistema distribuído, uma vez que apenas se utilizará uma máquina para a simulação dos sistemas. Esta componente implica também no conhecimento e investigação complementares, pelo que se apresenta como uma limitação ao trabalho desenvolvido.

## 6.2 Implementação

Numa primeira fase, foram instalados todos os SGBD necessários para o desenvolvimento da avaliação experimental. De forma sequencial foi efetuada a instalação do SQL Server 2014, PostGreSQL, MongoDB e, por último, o Cassandra.

Após a instalação, começou-se por idealizar um modelo físico de dados, tendo por base o modelo conceptual de dados, apresentado anteriormente (Figura 4-3), capaz de dar resposta ao sistema idealizado, de forma eficaz. O modelo desenvolvido que irá servir de base para o nosso sistema relacional único pode ser visualizado na Figura 6-1. Com base neste modelo de dados, foram criadas as bases de dados de teste *TeseMestradoSampleDB1*, *TeseMestradoSampleDB2* e *TeseMestradoSampleDB3* com o objetivo de armazenarem cinquenta mil, um milhão e cinco milhões de registos, respetivamente.

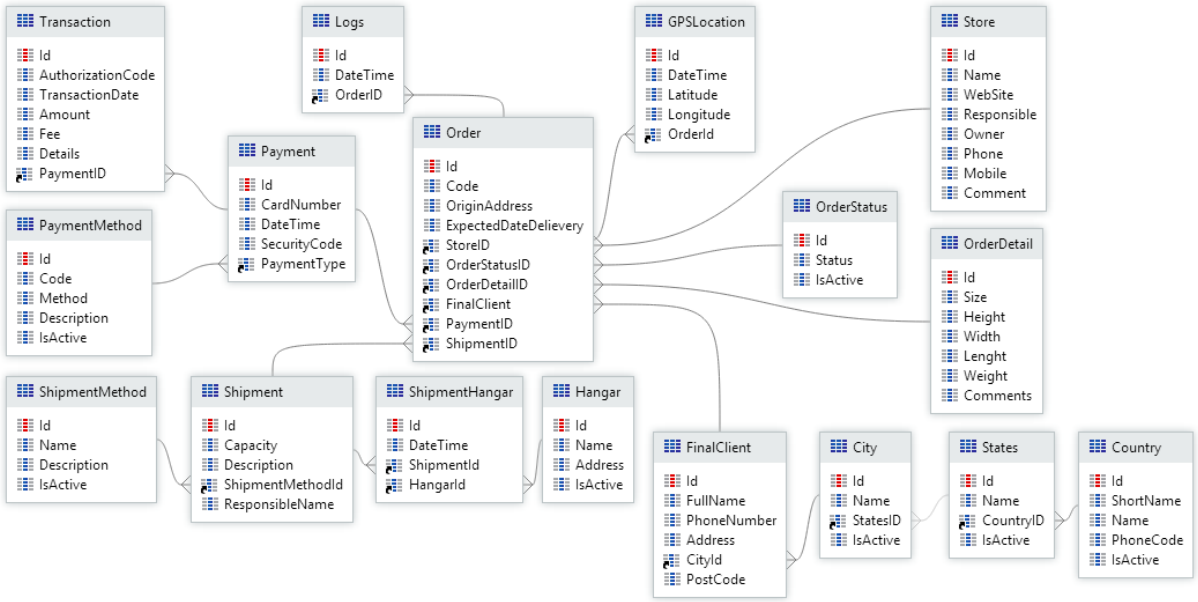



**Figura 6-1: Modelo e base de dados do sistema relacional único *TeseMestradoSampleDB1***

Como se pode observar, este sistema é composto por dezassete tabelas que têm como objetivo armazenar os dados relativos a encomendas e respetivos detalhes, pagamentos, clientes finais, entregas e *logs*. As encomendas são armazenadas na tabela *Order*. Cada encomenda está assim associada às suas características (*OrderDetail*), ao estado que se encontra (*OrderStatus*), à loja onde é oriunda (*Store*), ao cliente final que contém também o destino da encomenda (*FinalClient*), aos *logs* de acesso e procura (*Logs*), ao respetivo pagamento (*Payment*), transporte da encomenda (*Shipment*) e, por fim, à localização via GPS da encomenda, quando está junto da morada de destino (*GPSLocation*).

A tabela de pagamentos (*Payment*) está relacionada com as tabelas que indicam o método de pagamento (*PaymentMethod*) e a correspondente transação final (*Transaction*). Quanto à tabela relativa aos transportes de encomendas (*Shipment*) relaciona-se com as tabelas que armazenam a informação do método de envio utilizado (*ShipmentMehod*), ao registo do percurso do transporte tendo por base os armazéns (*ShipmentHangar*) e à informação de cada armazém (*Hangar*). Por último, há as tabelas relativas às cidades (*City*), estados (*States*) e países (*Country*) associadas à tabela dos clientes finais, de forma a completar a morada do cliente.

Após o sistema único estar desenvolvido, iniciou-se o desenvolvimento da segunda componente desta prova de conceito. Tendo como referência o modelo representado na Figura 4-3, foram então implementadas as estruturas de armazenamento de dados adequadas a utilizar no sistema poliglota.

O primeiro motor de base de dados instalado, presente no sistema poliglota, foi o PostGreSQL. Tendo por base a descrição do problema, foram construídas as bases de dados, *FinancialOrdersDB1*, *FinancialOrdersDB2* e *FinancialOrdersDB3* que serão povoadas por respetivamente por cinquenta mil, um milhão e cinco milhões de registos. A Figura 6-2 apresenta assim um exemplo (*FinancialOrdersDB1*) das bases de dados desenvolvidas para o armazenamento dos dados propostos.

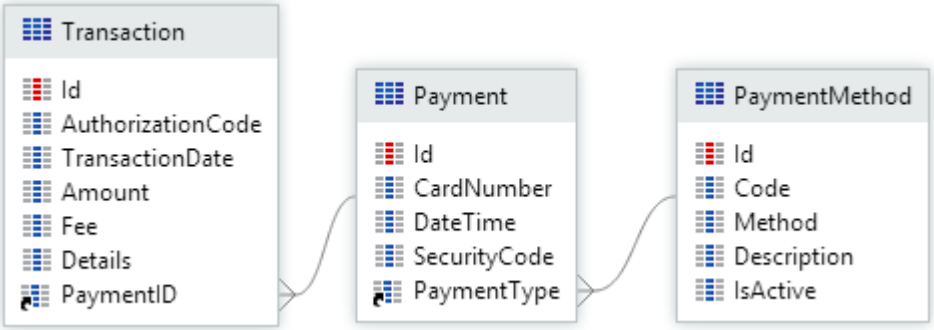



**Figura 6-2: Modelo de dados relativo à base de dados *FinancialOrdersDB1***

A base de dados *FinancialOrdersDB1* é assim composta pelas tabelas de pagamentos (*Payment*), métodos de pagamento possíveis (*PaymentMethod*) e, por último, pela a tabela das transações efetuadas (*Transaction*). Estas, têm a particularidade de estarem estruturadas e relacionadas de forma igual ao sistema que contém uma base de dados única.

De seguida, foi inicializada a implementação do trabalho prático relativo ao motor de base de dados não-relacional, MongoDB. De forma a simplificar a gestão dos dados, foi utilizada a ferramenta Robo 3T 1.1.1 (RoboMongo, 2017), que permite trabalhar sobre uma interface gráfica, não disponibilizada nativamente pela ferramenta MongoDB.

Com isto, foi construída a base de dados *OrdersTeseMestrado* assim como as respetivas coleções de dados *OrderInformationsDB1*, *OrderInformationsDB2* e *OrderInformationsDB3* que permitem o armazenamento dos dados respetivos aos diferentes volumes. A Figura 6-3 mostra a estrutura e os respetivos tipos de um documento da coleção criada.

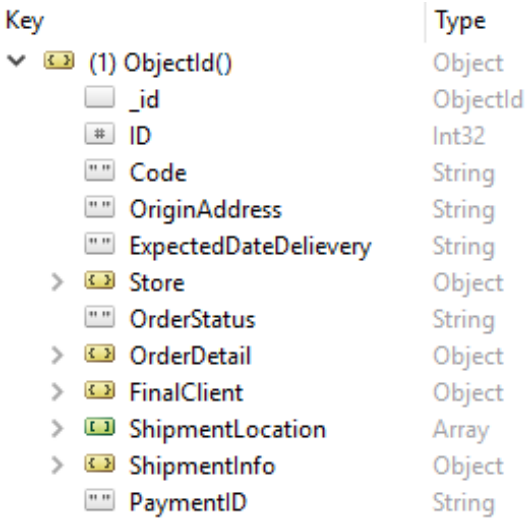



**Figura 6-3: Documento exemplo pertencente à coleção *OrderInformationsDB1***

Olhando a figura, percebe-se que a estruturação das tabelas relativas às encomendas foi totalmente reduzida a uma estrutura JSON que contém toda a informação pertinente. O campo *_id* identifica inequivocamente um documento (da coleção), atribuído através da função *ObjectID()* apresentada no topo da estrutura e que poderá substituir o campo *ID*. No entanto, é possível observar que este atributo foi mantido uma vez que é o identificador único da encomenda (*ID*), assim como se manteve o identificador único do pagamento (*PaymentID*), de forma a se manter a coerência e ligação entre os dados refletidos no SQL Server 2014.

Com isto, o MongoDB torna-se uma "ponte de ligação" entre as diferentes bases de dados que compõem o sistema poliglota. Tal, deve-se ao facto de se armazenarem os identificadores relativos ao pagamento das encomendas (*PaymentID*s), assim como, disponibilizar, o identificador único da encomenda (*ID*), às famílias de colunas de *logs* e de coordenadas GPS (campo *OrderId*), como se verá, posteriormente.

Por último, foi implementada a etapa relativa ao Cassandra. Aqui, foram primeiramente criados os *keyspaces* *TeseMestradoDB1*, *TeseMestradoDB2* e *TeseMestradoDB3* de modo a se armazenarem os três volumes de dados, respetivamente. Utilizando a linha de comandos que permite comunicar com o servidor de base de dados, pode-se exemplificar a criação do primeiro *keyspace* criado, através do seguinte comando em CQL:

```
CREATE KEYSPACE TeseMestradoDB1 WITH
     Replication={ 'Class' : 'SimpleStrategy', 'Replication_Factor' : 1 };
```

Este comando, como se pode verificar, contém duas particularidades relativas à replicação dos dados. A primeira é referente à estratégia de replicação *SimpleStrategy*, que consiste no método de replicação, aplicada aos *keyspaces* criados. A segunda é referente ao *Replication_Factor*, que apresenta o fator de replicação (número de réplicas a efetuar sobre os nós do *cluster*), neste caso, de um. De forma resumida, o *keyspace* criado será replicado apenas uma vez, tendo por base o método *SimpleStrategy*, dado que é utilizada uma só máquina. Posteriormente, foram criadas as famílias de colunas que têm como objetivo armazenar os dados relativos aos *logs* e às coordenadas de GPS, através dos seguintes comandos CQL:

```
CREATE TABLE Logs
(OrderId int, DateTime text, OrderCode text,
     PRIMARY KEY (OrderId, DateTime))
WITH CLUSTERING ORDER BY (DateTime DESC);

CREATE TABLE GPSLocation
(OrderId int, DateTime text, Latitude text, Longitude text, OrderCode text,
     PRIMARY KEY (OrderCode, DateTime))
WITH CLUSTERING ORDER BY (DateTime DESC);
```

A sintaxe da criação da família de colunas é bastante idêntica à do SQL, sendo apresentada, inicialmente, a tabela a criar, os campos e, posteriormente, as chaves primárias. Contudo, existem algumas particularidades desta linguagem, tais como a definição da chave primária e a ordem como são armazenados os dados.

A definição da chave primária, neste caso específico, é composta pela *Partition Key* e pela *Clustering Column*. A primeira define a forma como são armazenados os dados ao longo do *cluster*, ou seja, os campos com o mesmo *OrderId* e *OrderCode* serão armazenados no mesmo nó, no que diz respeito à família de colunas *Logs* e *GPSLocation*, respetivamente. As colunas que representam as chaves primárias foram definidas devido à forma como se irá consultar a

informação posteriormente, ou seja, serão estas as colunas que estarão presentes nas condições de filtragem da informação.

A segunda propriedade determina por que campo serão ordenados os dados, durante a inserção. Através do comando de *CLUSTERING* garante-se que a informação é armazenada de forma descendente, conforme a data e tempo de inserção, facilitando as consultas, posteriormente, neste sentido (Harrison, 2015). De modo automático, foram criados os índices relativos às colunas presentes na chave primária.

Concluindo, após a instalação, configuração, estruturação e construção dos motores de base de dados e das respetivas estruturas de dados, foram gerados e importados os dados.

## 6.3 Geração de Dados

Os dados a povoar os dois sistemas propostos, foram gerados. Como já referido atrás, a razão pela qual se optou por esta alternativa, face à adoção de um *dataset* real, ou seja, à utilização de uma coleção de dados referente a um SGBD em produção, deve-se à não permissão da partilha de um sistema e dos respetivos dados, pela grande maioria das organizações.

Desta forma, e visto que foram concebidas várias bases de dados específicas a cada problema a tratar na componente prática desta dissertação, tomou-se como obrigatória a geração de registos que correspondam aos modelos propostos. Os volumes a terem-se em consideração são de cinquenta mil, um milhão e cinco milhões de registos.

Inicialmente, foi gerado o primeiro volume de dados, de cinquenta mil registos, através da ferramenta Mockaroo, de modo a corresponder à base de dados única representada pela Figura 6-1. Estes registos, têm por base a geração de dados fictícios, mas que, no entanto, apresentam uma aparência realista, conforme o tipo de dados pretendido. Esta abordagem é utilizada por diversas empresas de forma a aplicarem os dados gerados, como dados de teste e, assim testarem as suas plataformas, antes da entrada em produção (Mockaroo, 2017).

Após a geração do primeiro volume de dados, estes foram integrados num espaço pessoal e grátis, atribuído a cada programador, da versão 10 da plataforma OutSystems. A utilização desta plataforma deveu-se à simplicidade, facilidade e rapidez na geração de dados com o objetivo de povoar as diferentes bases de dados. A OutSystems, de forma resumida, é uma plataforma

RAD (*Rapid Application Delivery*) de desenvolvimento *low-code*, classificada como um serviço PaaS, destinada ao desenvolvimento e entrega de aplicações *web* e móveis corporativas armazenadas na *cloud* (OutSystems, 2017).

Desta forma, a plataforma OutSystems foi utilizada para se gerarem um e cinco milhões de registos, correspondentes ao segundo e terceiro volumes de dados, de modo a analisar-se o desempenho dos dois sistemas, para análise da correlação entre volume e tempo de consulta. Os dois volumes de dados foram gerados através de diversas ações OutSystems (análogas a funções em diversas linguagens de programação) que permitiram a criação de dados aleatórios, baseados no primeiro volume de dados de cinquenta mil registos. Após estarem gerados os diferentes volumes, relativos à base de dados única criada, foram exportados e integrados no SGBD SQL Server 2014.

Seguidamente, foram gerados e exportados os dados relativos ao PostGreSQL, uma vez que estes correspondem exatamente às estruturas de dados (*Payment*, *Transaction* e *PaymentMethod*), do modelo de dados único. De seguida, foi criada uma ação OutSystems que permitiu exportar os três volumes de dados numa estrutura JSON, sendo posteriormente integrados no SGBD MongoDB. Por fim, de modo semelhante (através de uma ação OutSystems), foram gerados e exportados os *scripts* CQL, com os três volumes de dados a importar para a BD relativa ao SGBD Cassandra.

Cada volume de dados, terá como objetivo povoar três sistemas totalmente iguais de modo a se obterem três ambientes de teste individuais compostos por cinquenta mil registos e outros dois com um e cinco milhões de registos, respetivamente.

## 6.4 Consultas

Os testes serão baseados em consultas que têm como objetivo a recolha de tempos de consulta para a análise de ambos os sistemas. Visto que o sistema poliglota está dividido em três vertentes que refletem a divisão entre a faturação, encomendas, *logs* e coordenadas GPS serão, portanto, apresentadas e executadas consultas de forma individual, assim como, de forma global. A apresentação das consultas, será elaborada paralelamente entre o sistema poliglota e o sistema relacional com um único motor de base de dados.

### 6.4.1 Consultas Individuais

Apesar de se ter como principal objetivo avaliar o sistema poliglota perante um sistema único, é também importante efetuar testes de carácter individual. Desta forma, serão apresentadas um conjunto de consultas individuais, suscetíveis a análise, tendo por base os SGBD escolhidos e os diferentes volumes de dados. Numa primeira abordagem pretende-se efetuar e comparar diversas consultas entre os SGBD MongoDB e o SQL Server 2014.

Um dos principais pontos descritos no problema da prova de conceito foi a apresentação do trajeto de uma encomenda, tendo por base os pontos estratégicos (*Hangars*) que esta percorreu. Desta forma, foi criada uma consulta que dado um código de encomenda é apresentado o respetivo trajeto. Num contexto real, esta consulta permitiria ao cliente final seguir a sua encomenda ao longo do tempo. A consulta a efetuar na ferramenta MongoDB é traduzida pelo seguinte comando JSON:

```
db.OrderInformationsDB1.find(
    {
        Code: 'RG922935CN'
    },
    {
        "ExpectedDateDelievery": 1,
        "OriginAddress": 1,
        "ShipmentInfo.Method": 1,
        "ShipmentInfo.ResponsibleName": 1,
        "ShipmentLocation": 1
    }
)
```

Resumidamente, este comando irá efetuar uma consulta à coleção de dados *OrderInformationsDB1*, onde irá tentar encontrar o/s documento/s com o código *RG922935CN*. Com isto, irá devolver os valores relativos aos campos definidos, ou seja,

*ExpectedDateDelievery*, *OriginAddress*, *ShipmentInfo.Method*, *ShipmentInfo.ResponsibleName*, *ShipmentLocation*.

De forma análoga, a consulta a efetuar no SGBD SQL Server 2014, pode traduzir-se no comando de linguagem de consulta estruturada da seguinte forma:

```
SELECT [Order].ExpectedDateDelievery, [Order].OriginAddress,
     ShipmentMethod.Name, Shipment.ResponsibleName, ShipmentHangar.DateTime,
     Hangar.Name, Hangar.Address
FROM [Order]
INNER JOIN Shipment ON Shipment.ID=[Order].ShipmentId
INNER JOIN ShipmentMethod ON ShipmentMethod.ID=Shipment.ShipmentMethodId
INNER JOIN ShipmentHangar ON ShipmentHangar.ShipmentId=Shipment.Id
INNER JOIN Hangar ON Hangar.ID=ShipmentHangar.HangarId
WHERE [Order].Code='RG922935CN'
```

É possível observar que para processar a resposta a esta consulta é necessário aceder a cinco tabelas, quando utilizado o SQL Server. Com o MongoDB, como toda a informação é armazenada em documentos, apenas é necessário colocar a condição relativa ao código da encomenda e identificar os campos a devolver, de forma a apresentar o documento que contém a informação pretendida.

Ainda relativamente ao SGBD MongoDB, pretende-se efetuar uma consulta que apresente os dados relativos a um cliente, loja, características e estado de uma encomenda, dada uma morada e o respetivo código postal. Num contexto real, esta consulta permitiria identificar uma encomenda perdida, isto é, no caso de perda ou entrega incorreta da mesma, será sempre possível encontrar o cliente respetivo, através da morada correspondente. Deste modo, é apresentada toda a informação relativa à origem, destino e características que tem como objetivo corresponder à encomenda perdida.

```
db.OrderInformationsDB1.find(
     {
          "FinalClient.Address.Road": /.*96 King Street.*/,
          "FinalClient.Address.PostCode": /.*462-583-834.*/
     },
     {
          "Code": 1,
          "OriginAddress": 1,
          "OrderStatus": 1,
          "Store.Name":1,
          "Store.WebSite":1,
          "Store.Responsible":1,
          "Store.Mobile":1,
          "Store.Comment":1,
          "OrderDetail": 1,
          "FinalClient": 1
     })
```

Novamente, de forma resumida, este comando, pretende encontrar na coleção de dados *OrderInformationsDB1* toda a informação que corresponde às condições apresentadas inicialmente na estrutura JSON, ou seja, as condições *"FinalClient.Address.Road": /.*96 King Street.*/* e *"FinalClient.Address.PostCode": /.*462-583-834.*/*. Deste modo, será apresentada toda a informação relativa aos campos *Code*, *OriginAddress*, *OrderStatus*, *Store.Name*, *Store.WebSite*, *Store.Responsible*, *Store.Mobile* e *Store.Comment* e às estruturas correspondes ao *OrderDetail* e *FinalClient*.

De forma similar, pode-se traduzir a consulta anteriormente apresentada, no seguinte comando SQL, a realizar no motor de base de dados SQL Server 2014:

```
SELECT [Order].Code, [Order].OriginAddress, OrderStatus.Status,
    Store.Name, Store.WebSite, Store.Responsible, Store.Mobile,
    Store.Comment, OrderDetail.*, FinalClient.FullName,
    Country.PhoneCode, FinalClient.PhoneNumber, FinalClient.Address,
    FinalClient.PostCode, City.Name, States.Name, Country.Name
FROM [Order]
INNER JOIN Store ON Store.Id=[Order].StoreID
INNER JOIN OrderStatus ON OrderStatus.Id=[Order].OrderStatusID
INNER JOIN OrderDetail ON OrderDetail.Id=[Order].OrderDetailID
INNER JOIN FinalClient ON FinalClient.Id=[Order].FinalClient
INNER JOIN City ON City.Id=FinalClient.CityId
INNER JOIN States ON States.Id=City.StatesID
INNER JOIN Country ON Country.Id=States.CountryID
WHERE FinalClient.Address LIKE '%96 King Street%'
     and FinalClient.PostCode LIKE '%462-583-834%'
```

Comparando ambas as consultas ao nível estrutural constata-se que, mais uma vez, na perspetiva do MongoDB, apenas foi necessário colocar as condições necessárias para se filtrarem os dados presentes na coleção a apresentar. De forma paralela, para a consulta a elaborar no SGBDR SQL Server 2014, será necessário efetuar um conjunto de junções entre tabelas, de forma a devolver a informação esperada.

De seguida, numa segunda abordagem, será efetuada uma consulta recorrendo à base de dados Cassandra. Esta consulta tem como objetivo devolver a data e hora, assim como as coordenadas de GPS de uma encomenda, quando facultado o correspondente código de procura. Desta forma, será efetuada uma consulta à família de colunas relativa às coordenadas de GPS (*GPSLocation*). O comando CQL, relativo a esta BD, traduz-se da seguinte forma:

```
SELECT *
FROM GPSLocation
WHERE OrderCode='RG922935CN';
```

De forma análoga ao SQL Server 2014, pode-se constatar que o comando de consulta CQL é bastante idêntico. No entanto, para o motor de base de dados relacional é necessário efetuar uma junção com a tabela de encomendas e identificar a ordem com que se pretende devolver os valores. O comando SQL é assim traduzido da seguinte forma:

```
SELECT GPSLocation.* FROM GPSLocation
INNER JOIN [Order] ON [Order].Id= GPSLocation.OrderId
WHERE [Order].Code='RG922935CN'
ORDER BY GPSLocation.DateTime DESC
```

Por último, pode-se destacar que a consulta relativa ao motor de base de dados não-relacional Cassandra é, de certa forma, otimizada. Isto deve-se à adoção de apenas uma tabela onde está presente toda a informação, ao contrário do SQL Server 2014 onde é necessário efetuar uma junção. Paralelamente a esta constatação, pode-se também referir que as chaves primárias relativas ao Cassandra têm por base a *Partition Key* e a *Clustering Column* que permitem a ordenação dos dados de forma descendente, no momento da criação dos registos.

O último teste, passa pela consulta de dados ao motor de base de dados PostGreSQL. A consulta a realizar tem como objetivo obter todas as transações efetuadas, acompanhadas pelo método de pagamento escolhido, quando filtrado por um número de cartão de crédito específico. Num contexto real, esta consulta poderá mostrar ao utilizador todas as transações efetuadas a partir

de um dado cartão, por exemplo, em caso de extravio do mesmo, revertendo-as. A consulta SQL a efetuar na base de dados relativa à ferramenta PostGreSQL é a seguinte:

```
SELECT CardNumber, Method, DateTime, TransactionDate,
     SecurityCode, AuthorizationCode, Amount, Fee, Details
FROM Payment
INNER JOIN Transaction ON Transaction.PaymentID=Payment.Id
INNER JOIN PaymentMethod ON PaymentMethod.ID=Payment.PaymentType
WHERE Payment.CardNumber='10961120'
ORDER BY Payment.Id
```

A consulta correspondente a efetuar no SGBD SQL Server 2014, traduz-se através do seguinte comando:

```
SELECT CardNumber, Method, DateTime, TransactionDate,
     SecurityCode, AuthorizationCode, Amount, Fee, Details
FROM Payment
INNER JOIN [Transaction] ON [Transaction].PaymentID=Payment.Id
INNER JOIN PaymentMethod ON PaymentMethod.ID=Payment.PaymentType
WHERE Payment.CardNumber='10961120'
ORDER BY Payment.Id
```

Constata-se que ambas as consultas são idênticas devido ao facto de as bases de dados serem relacionais e, consequentemente, adotarem a mesma linguagem de consulta estruturada (SQL). Concluindo, todas as consultas apresentadas serão executadas nas diversas bases de dados, tendo em atenção o diferente volume de registos presentes nas mesmas, de modo a se analisarem os tempos de resposta às consultas efetuadas.

### 6.4.2 Consultas Globais

As consultas globais consistem na consulta instantânea e sequencial às bases de dados, por exemplo, caso seja necessário obter a informação presente no MongoDB e no Cassandra, de uma só vez, será obrigatório consultar-se as duas bases de dados, sequencialmente. Neste contexto, pretende-se estudar e analisar os tempos de resposta obtidos quando se consultam as diferentes bases de dados face ao sistema único.

Como se teve oportunidade de descrever atrás, o SGBD MongoDB será, de certa forma, a "ponte de ligação" entre as diferentes bases de dados que compõem o sistema poliglota. Em termos práticos, caso se deseje elaborar uma consulta que englobe os diferentes SGBD, será

necessário aceder aos valores armazenados no MongoDB, para que posteriormente estes sirvam de condições lógicas no momento da consulta do PostGreSQL ou do Cassandra, como se terá oportunidade de verificar.

Desta forma, para o primeiro teste das consultas globais, foi idealizada uma consulta que tem como objetivo a apresentação da informação completa sobre uma encomenda. Isto é, o seu estado e características, o pagamento e os termos do transporte da mesma, utilizando para o acesso à encomenda, um dado código de encomenda. Para se conseguir esta informação, será necessário efetuar um conjunto de consultas a diferentes SGBD.

Primeiramente, será consultada a base de dados MongoDB, de modo a se obterem todos os dados relativos à respetiva encomenda e, consequentemente, obter o identificador único do pagamento (*PaymentID*). Esta consulta, num contexto real, permite devolver ao utilizador a informação detalhada de uma encomenda. A primeira consulta a efetuar, neste conjunto de consultas é assim traduzida pelo seguinte comando:

```
db.OrderInformationsDB1.find(
      {
            "Code": 'RG922935CN',
      },
      {
            "OriginAddress": 1,
            "OrderStatus": 1,
            "Store": 1,
            "OrderDetail": 1,
            "FinalClient": 1,
            "ShipmentInfo": 1,
            "PaymentID": 1
      }
)
```

Tal como indicado anteriormente, será efetuada uma consulta JSON à coleção de dados *OrderInformationsDB1* através do campo relativo ao código da encomenda (*Code*), que neste caso em específico é representado pelo valor: *RG2922935CN*. Desta forma, pretende-se obter os valores correspondes à encomenda, apresentados através dos seguintes campos e estruturas: *OriginAddress*, *OrderStatus*, *PaymentID*, *Store*, *OrderDetail*, *FinalClient* e *ShipmentInfo*.

Obtidos os resultados da consulta realizada no motor de dados MongoDB, será efetuada uma consulta ao motor de base de dados PostGreSQL. A consulta terá por base o campo *PaymentID*

devolvido anteriormente, que servirá como condição para se obter a informação relativa à faturação da encomenda pretendida. O comando a efetuar traduz-se pela seguinte consulta SQL:

```
SELECT Method, CardNumber, DateTime, SecurityCode, AuthorizationCode,
     TransactionDate, Amount, Fee, Details
FROM Payment
INNER JOIN Transaction ON Transaction.PaymentID=Payment.Id
INNER JOIN PaymentMethod ON PaymentMethod.ID=Payment.PaymentType
WHERE Payment.Id= 32834
```

Neste caso em específico utilizou-se o identificador *32834* devolvido na primeira consulta desta abordagem, para se efetuar a condição lógica da consulta a executar no motor de base de dados PostGreSQL e, desta forma, obter os dados relativos à faturação da encomenda cujo o código é o *RG2922935CN*, como apresentado na consulta relativa ao motor de base de dados não relacional MongoDB.

De forma análoga, as duas consultas apresentadas correspondem ao seguinte comando SQL a efetuar sobre as bases de dados armazenadas no SGBDR SQL Server 2014:

```
SELECT OriginAddress, Status, Store.*, OrderDetail.*, FinalClient.*,
     ShipmentMethod.Name, ResponsibleName, Shipment.Description, Method,
     CardNumber, DateTime, SecurityCode, AuthorizationCode, TransactionDate,
     Amount, Fee, Details
FROM [Order]
INNER JOIN Store ON Store.Id=[Order].StoreID
INNER JOIN OrderStatus ON OrderStatus.Id=[Order].OrderStatusID
INNER JOIN OrderDetail ON OrderDetail.Id=[Order].OrderDetailID
INNER JOIN FinalClient ON FinalClient.Id=[Order].FinalClient
INNER JOIN City ON City.Id=FinalClient.CityId
INNER JOIN States ON States.Id=City.StatesID
INNER JOIN Country ON Country.Id=States.CountryID
INNER JOIN Shipment ON Shipment.Id=[Order].ShipmentID
INNER JOIN ShipmentMethod ON ShipmentMethod.Id=Shipment.ShipmentMethodId
INNER JOIN Payment ON Payment.Id=[Order].PaymentID
INNER JOIN [Transaction] ON [Transaction].PaymentID=Payment.Id
INNER JOIN PaymentMethod ON PaymentMethod.ID=Payment.PaymentType
WHERE [Order].Code='RG922935CN'
```

A segunda consulta, de âmbito geral, tem como objetivo devolver o número total de *logs* que o cliente final efetua na plataforma, ao procurar pela sua encomenda. Daqui compreende-se que, dado um código de procura, será devolvido o número total de *logs*, assim como a informação relativa ao cliente final. Num contexto real, esta consulta permitirá que um administrador visualize o número de vezes que o cliente final procurou pela sua encomenda e, desta forma,

por exemplo, reunir esforços para diminuir o tempo de entrega da mesma. Assim, será necessário, primeiramente, efetuar uma consulta à coleção de dados do SGBD MongoDB e, posteriormente, uma vez devolvido o identificador único relativo à encomenda (*ID*), efetuar a consulta no motor de base de dados Cassandra.

A consulta relativa à primeira base de dados é assim traduzida através do seguinte comando de procura:

```
db.OrderInformationsDB1.find(
      {
            "Code": 'RG922935CN',
      },
      {
            "ID": 1,
            "OriginAddress": 1,
            "FinalClient": 1,
      }
)
```

Esta consulta JSON irá efetuar uma consulta à coleção de dados *OrderInformationsDB1* com o objetivo de devolver a informação relativa ao *ID* e às estruturas *OriginAddress* e *FinalClient*, dado o seguinte código de encomenda: *RN922935CN*.

Devolvida a informação desejada será então utilizado o valor obtido na consulta anterior, relativo ao campo *ID*, para se efetuar a consulta CQL. A consulta a efetuar no SGBD Cassandra tem como objetivo apresentar o número de *logs* efetuados para o código de consulta, apresentado na consulta JSON acima referida. O comando CQL pode ser assim traduzido da seguinte forma:

```
SELECT COUNT(*)
FROM Logs
WHERE OrderId = 845;
```

Como é possível visualizar, neste caso em concreto, o identificador da encomenda devolvido pela consulta do MongoDB foi o *845*. Desta forma, foi executada uma consulta que se traduz na contagem de *logs* relativamente ao *OrderId* apresentado.

Estas duas consultas são assim agregadas num único comando SQL a aplicar sobre as bases de dados relativas ao sistema relacional único. A consulta SQL é traduzida da seguinte forma:

```
SELECT FinalClient.FullName, FinalClient.PhoneNumber, FinalClient.Address,
     FinalClient.PostCode, City.Name, States.Name, Country.Name,
     COUNT(Logs.Id) AS NumberOfLogs
FROM [Order]
INNER JOIN Logs ON Logs.OrderId=[Order].Id
INNER JOIN FinalClient ON FinalClient.Id=[Order].FinalClient
INNER JOIN City ON City.Id=FinalClient.CityId
INNER JOIN States ON States.Id=City.StatesID
INNER JOIN Country ON Country.Id=States.CountryID
WHERE [Order].Code='RG922935CN'
GROUP BY FinalClient.FullName, FinalClient.PhoneNumber, FinalClient.Address,
     FinalClient.PostCode, City.Name, States.Name, Country.Name
```

As consultas apresentadas serão executadas nas diversas bases de dados, tendo em atenção o diferente volume de registos presentes nas mesmas, de modo a se poderem analisar os tempos de consulta recolhidos.

Em resumo, como foi atrás visto, percebe-se que a obtenção da informação desejada, no sistema poliglota, necessita do acesso a diferentes bases de dados e da aplicação de diferentes sintaxes de consulta. Para tal, nos dois testes experimentais propostos consultaram-se duas bases de dados. Por outro lado, no sistema relacional único, tal como esperado, apenas se efetuou uma única consulta SQL, na base de dados correspondente ao SGBDR SQL Server 2014.

Concluindo, uma vez que se consultaram duas bases de dados diferentes, será necessário recolher e somar tempos de cada consulta. Assim, é possível efetuar uma comparação do tempo total da abordagem poliglota com o tempo relativo à consulta do sistema relacional único.

## 6.5 Resultados Obtidos

No seguimento da secção anterior, serão aqui apresentados os resultados obtidos. Estes resultados têm por base as médias obtidas em dez amostras sobre a mesma consulta, aplicados aos três diferentes volumes de dados.

Pode-se desde já salientar que a comparação efetuada apenas se aplica a este caso em específico, pelo que não se pretende generalizar os resultados obtidos noutros tipos de cenários, tais como na adoção de outros volumes de dados, motores de base de dados ou configurações no servidor ao nível de *software* e *hardware* pelo que poderão apresentar resultados divergentes dos obtidos.

### 6.5.1 Resultados das Consultas Individuais

Os volumes de dados de cerca de cinquenta mil, um milhão e cinco milhões de registos são traduzidos através dos volumes Volume 1, Volume 2 e Volume 3, respetivamente, presentes nos gráficos apresentados de seguida.

Os resultados obtidos na primeira consulta, que tinha como objetivo apresentar os pontos estratégicos e, consequentemente, o percurso de uma encomenda, dado o seu respetivo código de procura, estão apresentados na Figura 6-4. As médias de tempos de consulta recolhidas na execução da primeira consulta individual no SGBD MongoDB foram de 0.046, 0.226 e 0.792 segundos para o Volume 1, Volume 2 e Volume 3, respetivamente. Quanto ao motor de base de dados SQL Server 2014, foram recolhidas as médias de tempo de 0.108, 0.412 e 1.258 segundos para o Volume 1, Volume 2 e Volume 3, respetivamente.

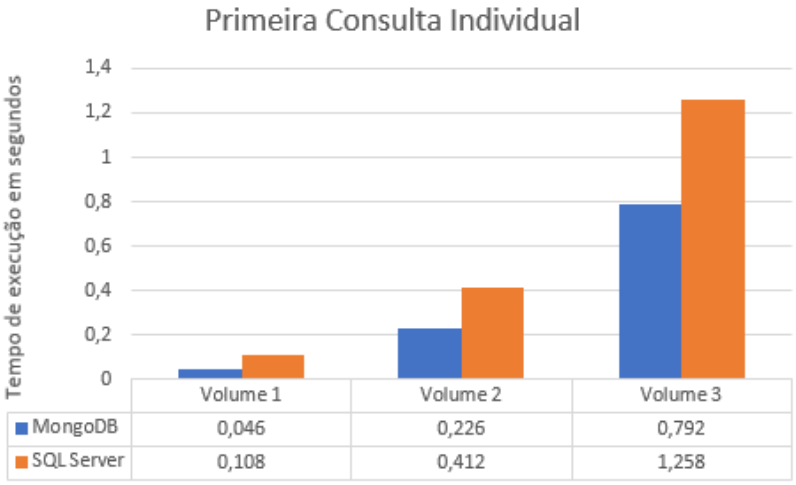



**Figura 6-4: Resultados obtidos na primeira consulta individual**

De modo a se obterem os resultados exatos no SGBD MongoDB, foi utilizada a função (nativa) *`explain("executionStats": 1)`*. Esta função tem como objetivo descriminar todos os processos que o MongoDB efetua para se obter a informação desejada, assim como apresentar o tempo de execução da consulta (MongoDB, 2017a). Por outro lado, para as consultas relativas ao SQL Server 2014 foi utilizada a ferramenta *Client Statistics*, nativa deste motor de base de dados, para se obterem os tempos de consulta efetuadas (Borl, 2012).

A segunda consulta individual, tinha como objetivo a apresentação da informação relativa ao cliente, loja, características e estado de uma encomenda, quando facultado, a respetiva morada e código postal, tendo-se obtido os resultados apresentados na Figura 6-5. Nesta consulta, a média de tempos recolhida para o motor de base de dados MongoDB foi de 0.178, 2.412 e

11.293 segundos, enquanto que para o SQL Server 2014 as médias foram de 0.247, 2.831 e 17.145 segundos para o Volume 1, Volume 2 e Volume 3, respetivamente.

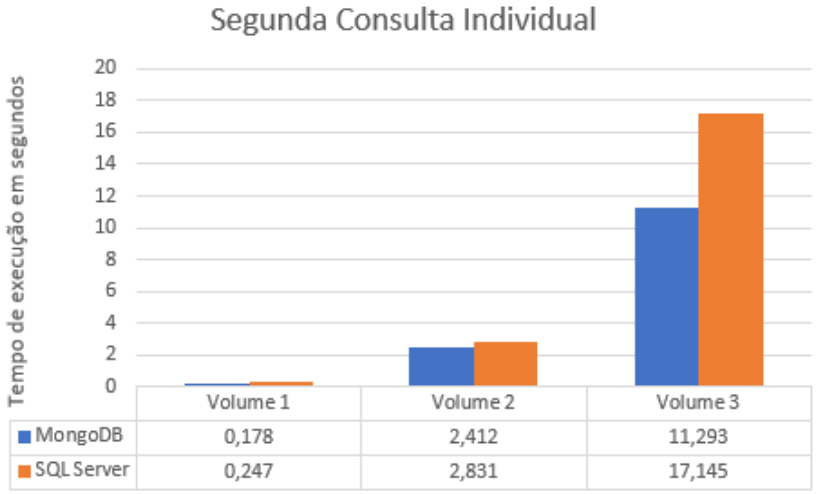



**Figura 6-5: Resultados obtidos na segunda consulta individual**

Tal como apresentado atrás, para se obterem os resultados visíveis na Figura 6-5, foram utilizadas os mesmos métodos que na consulta anterior, ou seja, a função (nativa) do MongoDB *`explain("executionStats": 1)`* e a ferramenta *Client Statistics* para o SQL Server 2014.

Relativamente à terceira consulta, que tem como objetivo devolver a data e hora, assim como as coordenadas de GPS de uma encomenda, quando facultado um código de procura, obtiveram-se os resultados apresentados pela Figura 6-6. Nesta consulta, como se pode observar, obtiveram-se as médias de 0.027, 0.055 e 0.258 segundos para o motor de base de dados Cassandra, enquanto que para o SQL Server obtiveram-se os valores médios de 0.069, 0.293 e 0.881 segundos, relativamente ao Volume 1, Volume 2 e Volume 3, respetivamente.

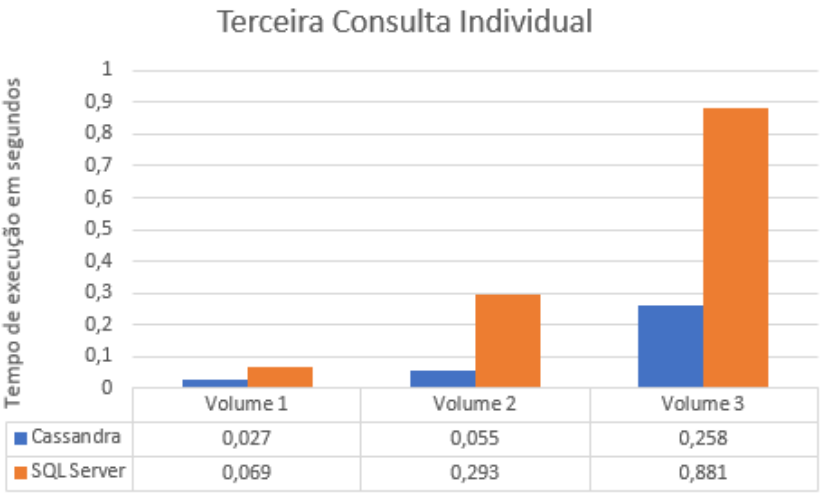



**Figura 6-6: Resultados obtidos na terceira consulta individual**

Quanto ao motor de base de dados Cassandra, utilizou-se o comando *tracing on;* para se obterem os tempos exatos de resposta à respetiva consulta. Com este comando, é apresentada uma lista descritiva de todos os processos efetuados para se processar a consulta CQL proposta, assim como o tempo de resposta, em milissegundos (DataStax, 2017). Relativamente ao SQL Server 2014, utilizou-se a ferramenta *Client Statistics*, como apresentado atrás.

A quarta e última consulta individual visa procurar todas as transações efetuadas, acompanhadas pelo método de pagamento escolhido, dado um número de cartão de crédito. Os resultados obtidos estão assim representados através da Figura 6-7. As médias obtidas com o motor de base de dados relacional PostGreSQL foram de 0.124, 0.253 e 0.749 segundos, enquanto que para o SQL Server 2014 foram de 0.205, 0.342 e 0.778 segundos para o Volume 1, Volume 2 e Volume 3, respetivamente.

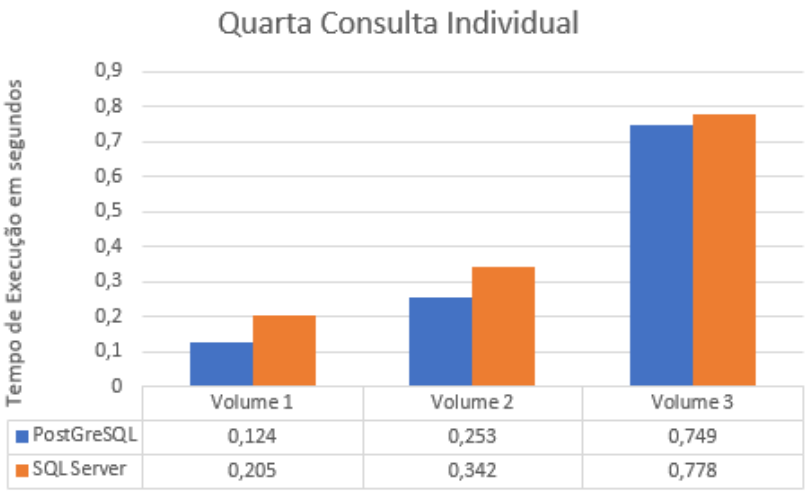



**Figura 6-7: Resultados obtidos na quarta consulta individual**

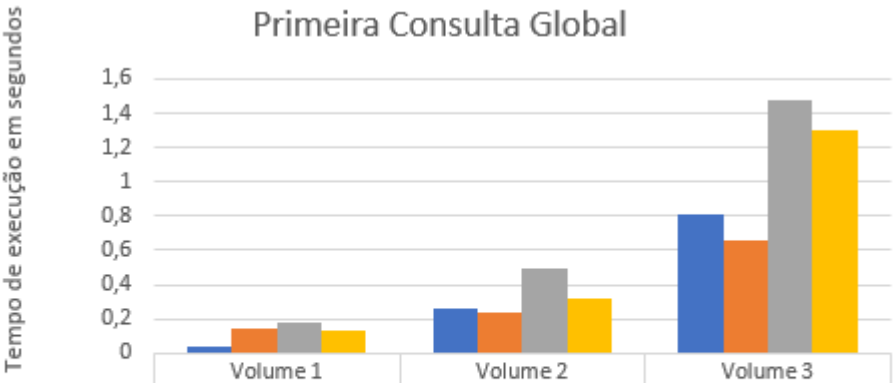
No motor de base de dados PostGreSQL utilizou-se o comando `EXPLAIN ANALYZE` para se obterem os resultados exatos de tempo de resposta à consulta efetuada, onde são apresentados todos os processos executados para se obter a informação desejada (PostgreSQL, 2017a). Relativamente ao SQL Server 2014, utilizou-se novamente a ferramenta *Client Statistics*.

### 6.5.2 Resultados das Consultas Globais

As consultas globais, tal como referido, têm como objetivo analisar os tempos de resposta de uma consulta, quando é necessário obter informação presente em diversos motores de base de dados, de uma só vez. Para se recolherem os tempos de consulta foram utilizados os mesmos métodos apresentados na secção anterior.

Desta forma, a primeira consulta global efetuada tinha como objetivo apresentar a informação completa sobre uma encomenda, ou seja, quando é dado um código de procura, correspondente à encomenda pretendida, seria apresentado o estado e características, o pagamento e os termos do transporte da mesma.

Os resultados obtidos para os diferentes motores de base de dados são apresentados pela Figura 6-8. Nesta consulta foram obtidas as médias de tempos de consulta de 0.043, 0.256 e 0.809 segundos para o MongoDB e de 0.138, 0.233 e 0.664 segundos para o PostGreSQL. Daqui, resultou a soma das médias dos os valores recolhidos anteriormente, de 0.181, 0.489 e 1.473 segundos, face às médias do motor de base de dados relacional de 0.134, 0.316 e 1.297 segundos, respetivamente ao Volume 1, Volume 2 e Volume 3.

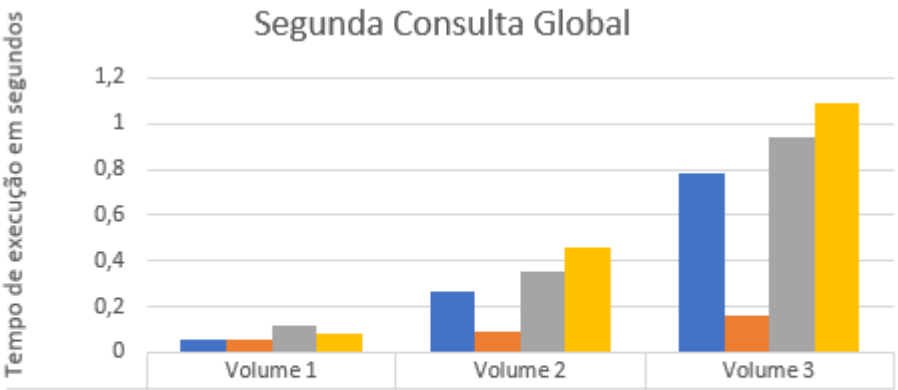



| | Volume 1 | Volume 2 | Volume 3 |
|---|---|---|---|
| MongoDB | 0,043 | 0,256 | 0,809 |
| PostGreSQL | 0,138 | 0,233 | 0,664 |
| Soma | 0,181 | 0,489 | 1,473 |
| SQL Server | 0,134 | 0,316 | 1,297 |

**Figura 6-8: Resultados obtidos na primeira consulta global**

A segunda e última consulta global tinha como objetivo devolver o número total de *logs* que o cliente final efetua na plataforma, ao procurar pela sua encomenda, ou seja, facultado um código de procura será devolvido o número total de *logs*, assim como a informação relativa ao cliente final. A Figura 6-9, abaixo apresentada, dita as médias de tempos obtidas nos diferentes SGBD.

As médias de tempos obtidas com a consulta relativa ao motor de base de dados MongoDB são de 0.055, 0.265 e 0.783 segundos, enquanto que, para o Cassandra foram de 0.058, 0.091 e de 0.158 segundos. Daqui, resultou a soma de médias relativas aos SGBD MongoDB e Cassandra de 0.113, 0.356 e 0.941 face aos resultados obtidos para o SQL Server 2014 que foram de 0.084, 0.456 e 1.089 segundos, respetivamente ao Volume 1, Volume 2 e Volume 3.




| | Volume 1 | Volume 2 | Volume 3 |
|---|---|---|---|
| MongoDB | 0,055 | 0,265 | 0,783 |
| Cassandra | 0,058 | 0,091 | 0,158 |
| Soma | 0,113 | 0,356 | 0,941 |
| SQL Server | 0,084 | 0,456 | 1,089 |

**Figura 6-9: Resultados obtidos na segunda consulta global**

## 6.6 Análise de Resultados

Depois de todos os procedimentos descritos nas secções anteriores desta dissertação foram, portanto, realizados os testes de desempenho às consultas apresentadas. Os resultados recolhidos basearam-se no tempo médio de resposta entre dez amostras recolhidas, executadas nos diversos motores de base de dados em estudo, assim como para os diferentes volumes de dados. Nesta secção serão, assim, analisados os resultados obtidos.

### 6.6.1 Análise de Resultados das Consultas Individuais

A primeira consulta efetuada teve como objetivo analisar e comparar o desempenho, quanto ao tempo de execução, relativamente aos SGBD MongoDB e SQL Server 2014. Os resultados, representados pela Figura 6-4, evidenciam uma clara diferença no desempenho dos motores de dados. O MongoDB apresentou valores médios de tempo de execução mais baixos (0.046, 0.226 e 0.792 segundos) que o SQL Server (0.108, 0.412 e 1.258 segundos), o que evidencia

um desempenho superior, por parte do primeiro. A explicação para este comportamento pode dever-se ao facto do SQL Server 2014 ter a seu cargo quatro junções, enquanto que no MongoDB, apenas é necessário efetuar a consulta através do código da encomenda e é devolvida a mesma informação correspondente, o que poderá causar um maior desempenho na consulta, sabendo-se que as junções são operações inerentemente "caras" (Parker, Poe e Vrbsky, 2013).

Na segunda consulta, também com os mesmos SGBD, é igualmente evidente um maior desempenho do motor de base de dados MongoDB (0.178, 2.412 e 11.293 segundos). Os resultados, que podem ser observados na Figura 6-5, evidenciam um maior tempo de execução no SQL Server 2014 (0.247, 2.831 e 17.145 segundos), apesar de não serem tão evidentes nos volumes 1 e 2, como na primeira consulta individual apresentada. Como referido, estes tempos podem ser justificados, por exemplo, através do grande número de junções que é necessário efetuar quando consultada a base de dados relacional.

Os resultados obtidos na terceira consulta individual são mostrados na Figura 6-6, onde são analisados os SGBD Cassandra e SQL Server 2014. As diferenças dos tempos de consulta são também evidentes entre os dois motores de base de dados. A média de tempos recolhidos para o Cassandra foram de 0.027, 0.055 e 0.258 segundos face aos 0.069, 0.293 e 0.881 segundos do SQL Server 2014. Desta forma, o primeiro apresenta um maior desempenho que o segundo, em mais de metade dos tempos recolhidos. Neste caso em específico, a explicação para o comportamento exibido pode advir da forma como a ferramenta Cassandra lida com os dados a armazenar. Ou seja, desde as propriedades que determinam a forma e o fator de replicação, até à criação das tabelas onde são definidas as chaves primárias tendo por base a *Clustering Column* e a *Partition Key*, terminando na forma como são armazenados e consultados de dados (unicamente através das chaves primárias), poderá justificar o desempenho superior face ao SQL Server 2014 (Cattell, 2011).

Por último, pode-se analisar os resultados obtidos na quarta consulta individual, onde é avaliado o desempenho entre os motores de base de dados PostGreSQL e SQL Server 2014. Os resultados obtidos são mostrados na Figura 6-7, apresentada na secção anterior. Como pode observar-se, a ferramenta relacional presente no sistema poliglota obtém (0.124, 0.253 e 0.749 segundos), também, um maior desempenho que o SQL Server 2014 (0.205, 0.342 e 0.778 segundos), apesar de não ser tão significativo como os tempos de execução praticados pelos

SGBD NoSQL. A diferença dos tempos de execução conforme os volumes de dados, evidenciam que a diferença dos tempos é mínima, sendo esta reduzida com o crescimento do volume de dados.

### 6.6.2 Análise de Resultados das Consultas Globais

As consultas globais tiveram como objetivo a análise dos tempos de execução conseguidos pelos diferentes motores de base de dados. A necessidade de apresentar informação presente em diferentes bases de dados poderá ser uma realidade, pelo que os tempos de execução destas consultas têm como propósito esclarecer se um sistema poliglota apresenta melhor desempenho que um sistema único.

A primeira consulta global permitiu analisar os tempos de execução obtidos nos SGBD MongoDB e PostGreSQL face ao motor SQL Server 2014. A média dos tempos obtidos pelos dois SGBD (0.043, 0.256 e 0.809 segundos para o MongoDB e de 0.138, 0.233 e 0.664 segundos para o PostGreSQL), apresentados na Figura 6-8, de forma individual, permitiu verificar que os tempos de consulta são inferiores ao sistema único relacional. Contudo, quando se efetua a soma dos mesmos, é possível concluir que o sistema poliglota não satisfaz as necessidades de se obter um sistema com melhores tempos de consulta, neste caso em específico, uma vez que a soma dos tempos de execução é superior (0.181, 0.489 e 1.473 segundos) ao tomado pela consulta relativa ao SQL Server 2014 (0.134, 0.316 e 1.297 segundos). Apesar do resultado não ser significantemente diferente, entre o sistema relacional e poliglota, representa um indício que efetuar consultas em diferentes motores de base de dados para se obter a informação desejada, poderá não ser a melhor abordagem a tomar. Esta conclusão é retirada com base no ligeiro aumento no tempo médio das consultas efetuadas no sistema poliglota perante o sistema único, que poderá continuar a aumentar de forma paralela ao volume de dados que uma dada aplicação armazena.

Por fim, a segunda consulta global, ao contrário da primeira, apresenta um aumento no desempenho quando o volume de dados sobe, no que diz respeito aos tempos médios de execução do sistema poliglota face ao sistema relacional único, como se verifica na Figura 6-9. Aqui, foram estudadas as médias de tempos de execução obtidos na execução das consultas nos SGBD MongoDB (055, 0.265 e 0.783 segundos) e Cassandra (0.058, 0.091 e de 0.158 segundos), relativos ao sistema poliglota *versus* a ferramenta SQL Server 2014. Após a soma dos resultados obtidos nos diferentes sistemas NoSQL (0.113, 0.356 e 0.941 segundos) face ao

sistema único (0.084, 0.456 e 1.089 segundos), pode-se afirmar que, perante o primeiro volume de dados, o sistema relacional único apresenta um maior desempenho que o sistema poliglota. No entanto, conforme o aumento de registos presentes nas bases de dados, o desempenho no SQL Server 2014 decai perante o conjunto de base de dados MongoDB - Cassandra, como evidencia a Figura 6-9.

### 6.6.3 Discussão Final

A elaboração de um trabalho desta envergadura permite compreender alguns dos requisitos fundamentais impostos por uma organização. A escolha e adoção dos SGBD a aplicar num sistema poliglota implicam tempo e recursos que muitas organizações não possuem para empreenderem num sistema deste tamanho.

O tempo inicial implicado na adoção de um sistema poliglota é descrito através da compreensão, análise e especificação de um problema. É também necessário ter em consideração a pesquisa e caracterização dos SGBD a adotar em cada divisão do problema e, posteriormente, a sua implementação. Daqui, resulta o problema dos recursos humanos presentes numa organização. Uma vez que a adoção de um sistema poliglota inclui a presença de diversos SGBD, será também necessário ter recursos qualificados para o efeito. Cada SGBD a adotar terá características, pormenores, especificações e princípios de que apenas os trabalhadores muito qualificados conseguirão tirar o melhor partido, ao aplicá-los num contexto real. A formação e qualificação de um profissional poderá traduzir-se num grande investimento que, por vezes, não é compensado com o retorno que depois se obtém.

Através da análise de resultados é possível perceber que o trabalho implicado na construção de um sistema poliglota poderá não compensar em sistemas onde se estima ter um baixo volume de dados. A diferença de tempos apresentados em todas as consultas, por exemplo, para o Volume 1, é de certa forma tão pequena que o investimento no desenvolvimento de um sistema que implemente o conceito de persistência poliglota poderá não traduzir o tempo e recursos gastos. Tal como no caso em estudado e tendo por base os casos de teste aplicados. Contudo, para sistemas complexos e com grande volume de dados (representado pelo Volume 3), a abordagem poliglota traduz um aumento de desempenho significativo para o seu sistema, que poderá justificar um grande investimento, como foi concluído para os resultados obtidos na Figura 6-9.

As consultas individuais apresentadas, neste problema prático, permitiram verificar que o tempo de execução tomado é inferior para o sistema poliglota, relativamente ao sistema único. Quanto às consultas globais apresentadas esta constatação não é certa, uma vez que os resultados obtidos no sistema poliglota não apresentam uma clara vantagem perante o sistema único, só o sendo para a segunda consulta global. Desta forma, é possível constatar que a individualização dos SGBD é a melhor abordagem a seguir, em alternativa às consultas que incluem diferentes motores de bases de dados. Contudo, esta afirmação carece de uma análise mais aprofundada, com mais consultas e testes, visto que, como foi analisado, o desempenho poderá variar conforme a quantidade de dados a armazenar.

Por fim, pode-se concluir que a organização dos dados é também uma mais-valia, no que diz respeito ao seu tipo. Adotar uma ferramenta que não foi idealizada para armazenar determinado tipo de dados poderá afetar negativamente o desempenho de um sistema, pelo que, e novamente, a escolha dos SGBD deverá ser ponderada de modo a dar a melhor resposta, perante o tipo de problema em questão.

# 7. Conclusão

O bom desempenho da base de dados é um requisito que qualquer organização pretende para as suas aplicações. O aumento de dados estruturados, semiestruturados e não-estruturados, traduzido pelo conceito de *Big Data*, evidenciou as debilidades dos sistemas tradicionais em processar e armazenar os mais diversos tipos de dados.

Apesar dos sistemas relacionais serem, nos dias de hoje, amplamente utilizados, outros tipos de sistemas surgiram para complementar o modelo relacional. Os sistemas NoSQL apresentam como vantagens a ausência de esquema das bases de dados, a escalabilidade horizontal, os sistemas distribuídos e os mecanismos de processamento de dados específicos de cada SGBD. No entanto, como uma só base de dados muitas das vezes não satisfaz os requisitos complexos das organizações, surge o conceito de Persistência Poliglota, que tem como objetivo integrar diversos tipos de base de dados, aproveitando o melhor de cada uma.

De modo a averiguar se os sistemas poliglotas são efetivamente uma alternativa viável face aos sistemas com uma única base de dados, foram desenvolvidos dois sistemas, um único e outro poliglota de modo a simular uma aplicação de entrega de encomendas. Numa primeira abordagem, adotou-se o SGBDR SQL Server 2014 para se simular o sistema único, enquanto que na segunda foram adotados os SGBD MongoDB, PostGreSQL e Cassandra para se armazenar os dados relativos às encomendas, às faturas e aos *logs* e coordenadas GPS, respetivamente. Neste seguimento, foi necessário analisar qual o SGBD a adotar, para cada problema em específico, tendo por base os tipos de dados, os princípios e características e se correspondem ao problema a tratar, o que tornou o processo de escolha bastante demorado.

Desta forma, foram idealizadas consultas individuais e globais, de forma a testar ambos os sistemas através de três volumes de dados e segundo a métrica de avaliação do desempenho relativa aos tempos de consulta. Os resultados obtidos evidenciaram que as consultas individuais, no sistema poliglota, apresentam um maior desempenho face ao sistema relacional. Por outro lado, quanto às consultas globais, é possível notar um ligeiro aumento do desempenho em volumes de dados elevados. No entanto, esta constatação carece de um maior número de testes, uma vez que os resultados não foram consistentes.

Em jeito de reflexão, pode afirmar-se que os objetivos propostos foram alcançados com sucesso. Conclui-se que, para a elaboração de um sistema poliglota, é necessária uma grande investigação sobre os SGBD a serem utilizados, assim como um investimento elevado, em termos de tempo e financeiros, na formação qualificada dos trabalhadores para a gestão futura do sistema. No entanto, esta abordagem poderá compensar, a médio-longo prazo, ao nível de desempenho, no que diz respeito ao tempo de execução de uma consulta, conforme o crescimento dos dados a armazenar, como foi verificado na análise de resultados obtidos.

## 7.1 Limitações

A primeira grande limitação encontrada foi na procura de um caso real para se aplicar na prova de conceito, uma vez que se teve de optar por uma alternativa que não é a ideal. Nesta limitação em particular, pode-se afirmar que o maior problema encontrado e imposto por esta abordagem foi na geração de dados fictícios com um aspeto real. A dificuldade em gerar volumes de dados como os apresentados nesta dissertação é alta, pelo que no final foram apresentadas algumas incoerências e falhas, tais como a repetição de registos e a apresentação de linhas que não correspondiam às chaves estrangeiras indicadas.

Outra limitação encontrada é relativa à extração dos resultados dos tempos de consulta. Isto deve-se ao facto de as consultas serem testadas diversas vezes sob diversos contextos que poderão ter afetado os resultados. Ou seja, devido ao facto de cada motor de base de dados apresentar técnicas e serviços nativos, que permitem o armazenamento dos resultados das consultas em memória *cache*, este poderão ter influenciado a recolha dos tempos de consulta e, consequentemente, ter afetado resultados apresentados nesta dissertação.

Por último, as características da máquina utilizada para efetuar a prova de conceito poderão não corresponder a uma máquina dedicada, no que se traduz num possível aumento dos tempos de execução das consultas apresentadas. Do mesmo modo, poderão implicar alguma aleatoriedade nos valores obtidos, uma vez que alguns dos serviços, nativos do SGBD, poderão ter sido lançados durante a execuções de cada teste, implicando alguma deriva nos valores obtidos.

## 7.2 Desenvolvimentos Futuros

Todos os trabalhos podem ser objeto de melhorias e este não é exceção. Dado o tempo disponível, a ênfase foi colocada na análise dos modelos de dados existentes, na idealização de dois sistemas, um de uma base de dados só e outro poliglota, onde se inclui a análise e escolha de diversos motores de base de dados. No entanto, tudo isto poderá ser melhorado desde a análise e escolha das tecnologias até à criação, configuração e gestão das bases de dados escolhidas para cada problema a tratar.

Um dos desenvolvimentos futuros, a ser aplicado no contexto desta dissertação, poderá ser a implementação de uma abordagem de persistência poliglota numa aplicação real ao nível organizacional, de modo a aplicar os conhecimentos obtidos nesta dissertação, assim como, obter uma perspetiva diferente num ambiente de produção. Aqui, é esperada a utilização de componentes de *hardware* dedicados ao processamento e dados, assim como a implementação do sistema poliglota em vários servidores, para perceber realmente como se comportariam os SGBD num sistema distribuído.

Por último, desenvolvimento futuro passa pela implementação de uma *framework MVC*, tal como a proposta neste trabalho, por exemplo, que permita efetuar a interação entre uma aplicação e o sistema poliglota que contém os diferentes motores de bases de dados. Esta *framework* tem como objetivo modelar e manter a coerência dos dados. Desta forma poderá disponibilizar a informação necessária através de serviços *web Representational State Transfer* (REST) ou *Simple Object Access Protocol* (SOAP) e assim facilitar o acesso aos dados em questão, por parte do sistema que os consome. Estes serviços apresentam uma mais-valia para um sistema, uma vez que não é necessário configurar a conexão à base de dados, preocupando-se apenas com a consulta e tratamento dos dados recebidos. Resumindo, esta *framework* pretende tratar de toda a logística de modelação e disponibilização dos dados a um ou mais sistemas, sem que estes tenham de tratar da conexão e modelação dos dados.

# 7 – Conclusão

# REFERÊNCIAS


Abramova, Veronika, Jorge Bernardino, e Pedro Furtado. 2014. «Experimental Evaluation of NoSQL Databases». *International Journal of Database Management Systems* 6 (3):01-16. https://doi.org/10.5121/ijdms.2014.6301.

Ahamed, Athiq. 2016. «Benchmarking Top NoSQL Databases». https://doi.org/10.13140/RG.2.1.2276.6969.

Alipui, Gilbert, Claude Asamoah, Richard Barilla, Leigh Anne Clevenger, Alecia Copeland, Sam Elnagdy, Hugh Eng, et al. 2014. «An Agile Approach to Doctoral Research Dissertation». *Proceedings of Student-Faculty Research Day, CSIS, Pace University*, D2.

Anikin, Denis. 2016. «When and why I use an in-memory database or a traditional database management system». Medium. 20 de Dezembro de 2016. https://medium.com/@denisanikin/when-and-why-i-use-an-in-memory-database-or-a-traditional-database-management-system-5737f6d406b5.

Atikoglu, Berk, Yuehai Xu, Eitan Frachtenberg, Song Jiang, e Mike Paleczny. 2012. «Workload analysis of a large-scale key-value store». Em *ACM SIGMETRICS Performance Evaluation Review*, 40:53–64. ACM. http://dl.acm.org/citation.cfm?id=2254766.

Bell, Sharon. 2015. «6 Critical Web Application Performance Metrics to Consider». CDNetworks. 29 de Outubro de 2015. https://www.cdnetworks.com/en/news/6-critical-web-application-performance-metrics-to-consider/4257.

Berg, Kristi L., Tom Seymour, e Richa Goel. 2013. «History Of Databases». *International Journal of Management & Information Systems (Online)* 17 (1):29.

Bonnet, Laurent, Anne Laurent, Michel Sala, Benedicte Laurent, e Nicolas Sicard. 2011. «Reduce, You Say: What NoSQL Can Do for Data Aggregation and BI in Large Repositories». Em , 483–88. IEEE. https://doi.org/10.1109/DEXA.2011.71.

Borl, Jes Schultz. 2012. «SQL Server Management Studio: "Include Client Statistics" Button». *Brent Ozar Unlimited®* (blog). 6 de Dezembro de 2012. https://www.brentozar.com/archive/2012/12/sql-server-management-studio-include-client-statistics-button/.

Brewer, Eric. 2012. «CAP twelve years later: How the" rules" have changed». *Computer* 45 (2):23–29.

Brewer, Eric A. 2000. «Towards robust distributed systems». Em *PODC*. Vol. 7. http://awoc.wolski.fi/dlib/big-data/Brewer_podc_keynote_2000.pdf.

Broadwell, Peter M. 2004. *Response time as a performability metric for online services*. Computer Science Division, University of California. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2004/CSD-04-1324.pdf.

Bryden, James. 2017. «Guide to Wide-Column Stores – NoSQL Explained – NoSQL Guide Tips and tricks for developers using NoSQL databases». 2 de Maio de 2017. http://nosql-guide.com/nosql-databases-explained-wide-column-stores/.

Cadcim, Technologies. 2010. *Oracle 11G: With Pl/Sql Approach*. Pearson Education India.

Cassandra, Liem, e Georgios Petropulos. 2016. «The economic value of personal data for online platforms, firms and consumers». 14 de Janeiro de 2016. http://bruegel.org/2016/01/the-economic-value-of-personal-data-for-online-platforms-firms-and-consumers/.

Cattell, Rick. 2011. «Scalable SQL and NoSQL data stores». *Acm Sigmod Record* 39 (4):12–27.

Cattuto, Ciro, Marco Quaggiotto, André Panisson, e Alex Averbuch. 2013. «Time-varying social networks in a graph database: A neo4j use case». Em *First international workshop on graph data management experiences and systems*, 11. ACM.

Chebotko, A., A. Kashlev, e S. Lu. 2015. «A Big Data Modeling Methodology for Apache Cassandra». Em *2015 IEEE International Congress on Big Data*, 238–45. https://doi.org/10.1109/BigDataCongress.2015.41.

Chen, S., X. Tang, H. Wang, H. Zhao, e M. Guo. 2016. «Towards Scalable and Reliable In-Memory Storage System: A Case Study with Redis». Em *2016 IEEE Trustcom/BigDataSE/ISPA*, 1660–67. https://doi.org/10.1109/TrustCom.2016.0255.

Chickerur, S., A. Goudar, e A. Kinnerkar. 2015. «Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications». Em *2015 8th International Conference on Advanced Software Engineering Its Applications (ASEA)*, 41–47. https://doi.org/10.1109/ASEA.2015.19.

Codd, E. F. 1985a. «Is your DBMS Really Relational?» *Computerworld, Inc.*, 14 de Outubro de 1985.

———. 1985b. «Does your DBMS Run by the Rules?» *Computerworld, Inc.*, 21 de Outubro de 1985.

Codd, Edgar. 1970. «A relational model of data for large shared data banks». *Communications of the ACM* 13 (6):377–387.

Connolly, Thomas M., e Carolyn E. Begg. 2005. *Database systems: a practical approach to design, implementation, and management*. 4th ed. International computer science series. Harlow, Essex, England ; New York: Addison-Wesley.

Conrad, Tim. 2006. *Postgresql vs. mysql vs. commercial databases: It's all about what you need*. Technical report, Devx, 2004. cosmoglobecorp. com/pdf/PostgreSQL_vs_ MySQL_vs_DB2_vs_MSSQL_vs_Oracle. pdf. http://vja.es/Postgresql%20Vs%20Mysql%20Vs%20Db2%20Vs%20Mssql%20Vs%20Oracle.pdf.

Coronel, Carlos, e Steven Morris. 2017. *Database Systems: Design, Implementation, and Management*. 12.ª ed. Cengage Learning.

Costa, Rogério. 2007. *SQL Guia Prático - 2a edição*. Brasport.

CouchDB. 2017. «Apache CouchDB». 2017. http://couchdb.apache.org/.

Cunha, José Pedro. 2015. «Column-based databases: estudo exploratório no âmbito das bases de dados NoSQL». https://repositorium.sdum.uminho.pt/handle/1822/40079.

Damesha, Hardeep Singh. 2015. «Object Oriented Database Management Systems-Concepts, Advantages, Limitations and Comparative Study with Relational Database Management Systems». *Global Journal of Computer Science and Technology* 15 (3). http://www.computerresearch.org/index.php/computer/article/view/1169.

Danielsen, Asbjørn. 1998. «The Evolution Of Data Models And Approaches To Persistence In Database Systems». 6 de Maio de 1998. https://www.fing.edu.uy/inco/grupos/csi/esp/Cursos/cursos_act/2000/DAP_DisAvDB/documentacion/OO/Evol_DataModels.html.

DataStax. 2017. «TRACING | CQL for Cassandra 2.1 | CQL for Cassandra 2.1». 2017. https://docs.datastax.com/en/cql/3.1/cql/cql_reference/tracing_r.html.

Date, C. J. 2003. *An Introduction to Database Systems*. 8th Edition. Pearson.

Datt, Niteshwar. 2016. «Comparative Study of CouchDB and MongoDB – NoSQL Document Oriented Databases». *International Journal of Computer Applications* 136 (3):24–26. https://doi.org/10.5120/ijca2016908457.

Davidson, Edward. 1982. «Evaluating database management systems». Em *Proceedings of the June 7-10, 1982, national computer conference*, 639–648. ACM. http://dl.acm.org/citation.cfm?id=1500855.

DB-Engines. 2017. «DB-Engines Ranking - Popularity ranking of database management systems». DB-Engines Ranking - popularity ranking of database management systems. 2017. http://db-engines.com/en/ranking.

Deacon, John. 2009. «Model-view-controller (mvc) architecture». *Online][Citado em: 10 de março de 2006.] http://www. jdl. co. uk/briefings/MVC. pdf*.

Dean, Jeffrey, e Sanjay Ghemawat. 2004. «MapReduce: Simplified Data Processing on Large Clusters». Em *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, 10–10. OSDI'04. Berkeley, CA, USA: USENIX Association. http://dl.acm.org/citation.cfm?id=1251254.1251264.

DeCandia, Giuseppe, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, e Werner Vogels. 2007. «Dynamo: amazon's highly available key-value store». *ACM SIGOPS operating systems review* 41 (6):205–220.

Deutsche Post DHL Group. 2017. «2016 Annual Report - Constantly Reinventing the Future of Logistics».

Elmasri, Ramez, e Sham Navathe. 2011. *Fundamentals of database systems*. 6th ed. Boston: Addison-Wesley.

EndPoint. 2016. *Benchmarking Top NoSQL Databases*.

Erb, Benjamin. 2012. «Concurrent programming for scalable web architectures».

Fan, Wei, e Albert Bifet. 2013. «Mining big data: current status, and forecast to the future». *ACM sIGKDD Explorations Newsletter* 14 (2):1–5.

Fojtik, David, Petr Podesva, e Jan Gebauer. 2014. *2014 15th International Carpathian Control Conference (ICCC 2014): Velke Karlovice, Czech Republic, 28 - 30 May 2014*. Piscataway, NJ: IEEE.

Fowler, Martin. 2009. *Padrões de Arquitetura de Aplicações Corporativas*. Bookman.

Gandomi, Amir, e Murtaza Haider. 2015. «Beyond the Hype: Big Data Concepts, Methods, and Analytics». *International Journal of Information Management* 35 (2):137–44. https://doi.org/10.1016/j.ijinfomgt.2014.10.007.

Ghemawat, Sanjay, Howard Gobioff, e Shun-Tak Leung. 2003. «The Google file system». Em *ACM SIGOPS operating systems review*, 37:29–43. ACM.

Gilbert, Seth, e Nancy Lynch. 2002. «Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services». *Acm Sigact News* 33 (2):51–59.

———. 2012. «Perspectives on the CAP Theorem». *Computer* 45 (2):30–36. https://doi.org/10.1109/MC.2011.389.

Gopal, R. Chandangole, e A. Tidke Bharat. 2016. «A generic tool to process mongodb or Cassandra dataset using Hadoop streaming». Em *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on*, 273–276. IEEE. http://ieeexplore.ieee.org/abstract/document/7724270/.

Grolinger, Katarina, Wilson A. Higashino, Abhinav Tiwari, e Miriam AM Capretz. 2013. «Data Management in Cloud Environments: NoSQL and NewSQL Data Stores». *Journal of Cloud Computing: Advances, Systems and Applications* 2 (1):22. https://doi.org/10.1186/2192-113X-2-22.

Han, Jing, E. Haihong, Guan Le, e Jian Du. 2011. «Survey on NoSQL database». Em *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, 363–366. IEEE. http://ieeexplore.ieee.org/abstract/document/6106531/.

Harrison, Guy. 2015. *Next Generation Databases: NoSQLand Big Data*. Apress. https://books.google.pt/books?id=q6pPCwAAQBAJ.

Hecht, Robin, e Stefan Jablonski. 2011. «NoSQL evaluation: A use case oriented survey». Em *2011 International Conference on Cloud and Service Computing*, 336–41. IEEE. https://doi.org/10.1109/CSC.2011.6138544.

Hewitt, Eben. 2011. *Cassandra: The Definitive Guide: [Distributed Data at Web Scale]*. 1. ed. Beijing: O'Reilly.

Hill, Mark D. 1990. «What is scalability?» *ACM SIGARCH Computer Architecture News* 18 (4):18–21.

Hu, H., Y. Wen, T. S. Chua, e X. Li. 2014. «Toward Scalable Systems for Big Data Analytics: A Technology Tutorial». *IEEE Access* 2:652–87. https://doi.org/10.1109/ACCESS.2014.2332453.

IBM. 2012. «IBM100 - Information Management System». CTB14. 7 de Março de 2012. http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/ibmims/.

iDatalabs. 2017. «Database Management System products and their install base». 2017. https://idatalabs.com/tech/database-management-system.

Intel. 2013. «Getting-Started-with-Hadoop-Planning-Guide.pdf». Fevereiro de 2013. http://www.triforce.com.au/pdf/getting-started-with-hadoop-planning-guide.pdf.

Jiménez-Peris, R., M. Patiño-Martinez, I. Brondino, e V. Vianello. 2016. «Transactional Processing for Polyglot Persistence». Em *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 150–52. https://doi.org/10.1109/WAINA.2016.144.

Jorgensen, Adam, James Rowland-Jones, John Welch, Dan Clark, Christopher Price, e Brian Mitchell. 2014. *Microsoft Big Data Solutions*. John Wiley & Sons.

Kamal, Hamaz, e Benchikha Fouzia. 2017. «A novel method for providing relational databases with rich semantics and natural language processing». *Journal of Enterprise Information Management* 30 (3):503–25.

Kanoje, S., V. Powar, e D. Mukhopadhyay. 2015. «Using MongoDB for social networking website deciphering the pros and cons». Em *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, 1–3. https://doi.org/10.1109/ICIIECS.2015.7192924.

Kaur, Karamjit, e Rinkle Rani. 2013. «Modeling and querying data in NoSQL databases». Em *Big Data, 2013 IEEE International Conference on*, 1–7. IEEE. http://ieeexplore.ieee.org/abstract/document/6691765/.

———. 2015. «A smart polyglot solution for big data in healthcare». *IT Professional* 17 (6):48–55.

Kedar, Seema. 2007. *Database Management Systems*. 2.ª ed. Technical Publications.

Kemmis, Stephen, e Robin McTaggart. 2005. «Communicative action and the public sphere». *The Sage handbook of qualitative research* 3:559–603.

Keznikl, Jaroslav, Michal Malohlava, Tomas Bures, e Petr Hnetynka. 2011. «Extensible Polyglot Programming Support in Existing Component Frameworks». Em *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, 107–115. IEEE. http://ieeexplore.ieee.org/abstract/document/6068329/.

Kitchin, Rob. 2013. «Big data and human geography: Opportunities, challenges and risks». 10 de Dezembro de 2013. http://eprints.maynoothuniversity.ie/5366/1/RK_big%20data%20human.pdf.

Kriegel, Alex. 2011. *Discovering SQL: A Hands-On Guide for Beginners*. John Wiley & Sons.

Lakshman, Avinash, e Prashant Malik. 2009. «Cassandra: Structured Storage System on a P2P Network». Em *Proceedings of the 28th ACM Symposium on Principles of Distributed*

*Computing*, 5–5. PODC '09. New York, NY, USA: ACM. https://doi.org/10.1145/1582716.1582722.

———. 2010. «Cassandra: a decentralized structured storage system». *ACM SIGOPS Operating Systems Review* 44 (2):35–40.

Lamllari, Rilinda. 2013. «Extending a methodology for migration of the database layer to the cloud considering relational database schema migration to NoSQL».

Laney, Douglas. 2001. «3D Data Management: Controlling Data Volume, Velocity, and Variety». META Group Inc.

Leavitt, Neal. 2010. «Will NoSQL databases live up to their promise?» *Computer* 43 (2). http://ieeexplore.ieee.org/abstract/document/5410700/.

Leberknight, Scott. 2008. «Scott Leberknight's Weblog». Polyglot Persistence. 15 de Outubro de 2008. http://www.sleberknight.com/blog/sleberkn/entry/polyglot_persistence.

Maurer, Hermann, Nick Scherbakov, Zahran Halim, e Zaidah Razak. 1998. «Abstract Data Objects». Em *From Databases to Hypermedia*, 165–173. Springer. http://link.springer.com/chapter/10.1007/978-3-642-58763-4_12.

Ming Wu, Chieh. 2015. «Comparisons Between MongoDB and MS-SQL Databases on the TWC Website». *American Journal of Software Engineering and Applications* 4 (2):35. https://doi.org/10.11648/j.ajsea.20150402.12.

Mockaroo. 2017. «Mockaroo - Random Data Generator and API Mocking Tool | JSON / CSV / SQL / Excel». 2017. https://mockaroo.com/.

Mohamed, Mohamed A., Obay G. Altrafi, e Mohammed O. Ismail. 2014. «Relational vs. nosql databases: A survey». *International Journal of Computer and Information Technology* 3 (03):598–601.

MongoDB. 2017a. «Explain Results — MongoDB Manual 3.4». https://github.com/mongodb/docs/blob/v3.4/source/reference/explain-results.txt. 2017. https://docs.mongodb.com/manual/reference/explain-results/.

———. 2017b. «MongoDB for GIANT Ideas». MongoDB. 2017. https://www.mongodb.com/index.

MSG. 2017. «What are Metrics and Why are they Important ?» 2017. http://www.managementstudyguide.com/what-are-metrics.htm.

Murazza, M. R., e A. Nurwidyantoro. 2016. «Cassandra and SQL database comparison for near real-time Twitter data warehouse». Em *2016 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, 195–200. https://doi.org/10.1109/ISITIA.2016.7828657.

Nayak, Ameya, Anil Poriya, e Dikshay Poojary. 2013. «Type of NOSQL databases and its comparison with relational databases». *International Journal of Applied Information Systems* 5 (4):16–19.

Nielsen, Jakob. 1994. *Usability Engineering*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA ©1993. https://www.nngroup.com/articles/response-times-3-important-limits/.

Nielsen, Paul, Mike White, e Uttam Parui. 2009. *Microsoft SQL Server 2008 Bible*. Indianapolis, Ind: Wiley.

Nyati, Suyog S., Shivanand Pawar, e Rajesh Ingle. 2013. «Performance evaluation of unstructured NoSQL data over distributed framework». Em , 1623–27. IEEE. https://doi.org/10.1109/ICACCI.2013.6637424.

O'Reilly, T. 2009. *What is Web 2.0*. O'Reilly radar report. O'Reilly Media. https://books.google.pt/books?id=NpEk_WFCMdIC.

Ostrower, Jon. 2016. «Airlines report problems as booking system goes down». CNNMoney. 11 de Novembro de 2016. http://money.cnn.com/2016/11/11/news/airline-systems-outage/index.html.

OutSystems. 2017. «The #1 Low-Code Platform for Digital Transformation | OutSystems». 2017. http://www.outsystems.com/.

Parker, Zachary, Scott Poe, e Susan V. Vrbsky. 2013. «Comparing NoSQL MongoDB to an SQL DB». Em , 1. ACM Press. https://doi.org/10.1145/2498328.2500047.

Paul, Subharthi. 2008. «Database Systems Performance Evaluation Techniques». *St. Louis, MI, USA*, Janeiro. http://www.cs.wustl.edu/~jain/cse567-08/ftp/db/.

Pirk, Holger, Florian Funke, Martin Grund, Thomas Neumann, Ulf Leser, Stefan Manegold, Alfons Kemper, e Martin Kersten. 2013. «CPU and cache efficient management of memory-resident databases». Em *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, 14–25. IEEE. http://ieeexplore.ieee.org/abstract/document/6544810/.

PostgreSQL. 2017a. «PostgreSQL: Documentation: 9.1: EXPLAIN». 2017. https://www.postgresql.org/docs/9.1/static/sql-explain.html.

———. 2017b. «PostgreSQL: The world's most advanced open source database». 2017. https://www.postgresql.org/.

Powell, Gavin. 2006. *Beginning database design*. Indianapolis, IN: Wiley.

Prasad, Abhishek, e Bhavesh N. Gohil. 2014. «A Comparative Study of NoSQL Databases.» *International Journal of Advanced Research in Computer Science* 5 (5). http://www.ijarcs.in/index.php/Ijarcs/article/view/2183.

Pritchett, Dan. 2008. «Base: An acid alternative». *Queue* 6 (3):48–55.

Purcell, Bernice. 2012. «Emergence of "Big Data" technology and analytics». *Journal of Technology Research* 4.

Rabl, Tilmann, Sergio Gómez-Villamor, Mohammad Sadoghi, Victor Muntés-Mulero, Hans-Arno Jacobsen, e Serge Mankovskii. 2012. «Solving big data challenges for enterprise application performance management». *Proceedings of the VLDB Endowment* 5 (12):1724–1735.

Redis. 2017. «Redis». 2017. https://redis.io/.

Rhodes, Ron. 2004. *The 10 Things You Should Know About the Creation vs. Evolution Debate*. Harvest House Publishers.

Rizvi, Sanam Shahla, e Tae-Sun Chung. 2010. «Flash SSD vs HDD: High performance oriented modern embedded and multimedia storage systems». Em *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, 7:V7–297. IEEE. http://ieeexplore.ieee.org/abstract/document/5485421/.

Rizzatti, Lauro. 2016. «Digital Data Storage is Undergoing Mind-Boggling Growth | EE Times». EETimes. 14 de Setembro de 2016. http://www.eetimes.com/author.asp?section_id=36&doc_id=1330462.

Rob, Peter, e Carlos Coronel. 2009. *Database Systems - Design, Implementation, and Management*. Eighth Edition. United States: Thomson - Course Technology.

Robinson, Ian, Jim Webber, e Emil Eifrem. 2015. *Graph databases: new opportunities for connected data*. O'Reilly Media, Inc.

RoboMongo. 2017. «Robo 3T - formerly Robomongo — native MongoDB management tool (Admin UI)». 2017. https://robomongo.org/.

Sadalage, Pramod J., e Martin Fowler. 2012. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley.

———. 2013. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Upper Saddle River, NJ: Addison-Wesley.

Sagiroglu, Seref, e Duygu Sinanc. 2013. «Big data: A review». Em , 42–47. IEEE. https://doi.org/10.1109/CTS.2013.6567202.

Saino, L., I. Psaras, e G. Pavlou. 2016. «Understanding sharded caching systems». Em *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524442.

Schaarschmidt, Michael, Felix Gessert, e Norbert Ritter. 2015. «Towards Automated Polyglot Persistence.» Em *BTW*, 73–82. https://www.baqend.com/paper/btw-polyglot.pdf.

Scriptcase. 2013. «O que é a metodologia Scrum em projetos?» Scriptcase - Desenvolvimento, Web Design, Marketing Digital e Vendas. 29 de Maio de 2013.

Selfa, D.M., Maya Carrillo, e M Del Rocio Boone. 2006. «A Database and Web Application Based on MVC Architecture». Em *16th International Conference on Electronics, Communications and Computers (CONIELECOMP'06)*, 48–48. https://doi.org/10.1109/CONIELECOMP.2006.6.

Sendir, B., M. Govindaraju, R. Odaira, e P. Hofstee. 2016. «Optimized Durable Commitlog for Apache Cassandra Using CAPI-Flash». Em *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, 156–63. https://doi.org/10.1109/CLOUD.2016.0030.

Shankland, Stephen. 2008. «Amazon suffers U.S. outage on Friday». CNET. 6 de Junho de 2008. https://www.cnet.com/news/amazon-suffers-u-s-outage-on-friday/.

Shuen, Amy. 2008. *Web 2.0: A Strategy Guide: Business Thinking and Strategies behind Successful Web 2.0 Implementations.* O'Reilly Media, Inc.

Silberschatz, Abraham, Henry F. Korth, e S. Sudarshan. 2011. *Database system concepts*. 6th ed. New York: McGraw-Hill.

Silva, Carlos André Reis Fernandes Oliveira. 2011. «Data modeling with NoSQL: How, when and why», Julho. https://repositorio-aberto.up.pt/bitstream/10216/61586/1/000148158.pdf.

Smith, Lisa. 2015. «What PostgreSQL has over other open source SQL databases: Part I». Compose Articles. 8 de Outubro de 2015. https://www.compose.com/articles/what-postgresql-has-over-other-open-source-sql-databases/.

Srivastava, Kriti, e Narendra Shekokar. 2016. «A Polyglot Persistence approach for E-Commerce business model». Em *Information Science (ICIS), International Conference on*, 7–11. IEEE. http://ieeexplore.ieee.org/abstract/document/7845291/.

Stephens, Rod. 2010. *Beginning Database Design Solutions*. John Wiley & Sons.

Strauch, Christof. 2011. «NoSQL databases». *Lecture Selected Topics on Software-Technology Ultra-Large Scale Sites - Stuttgart Media University*, 149.

Tezer, O. S. 2014. «SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems». DigitalOcean. 21 de Fevereiro de 2014. https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems.

The Economist. 2017. «The world's most valuable resource is no longer oil, but data». The Economist. 5 de Junho de 2017. http://www.economist.com/news/leaders/21721656-data-economy-demands-new-approach-antitrust-rules-worlds-most-valuable-resource.

Tiwari, Shashank. 2011. *Professional NoSQL*. Indianapolis, Ind: Wiley.

Toth, Renato Molina. 2011. «Abordagem NoSQL–uma real alternativa». *Sorocaba, São Paulo, Brasil: Abril* 13. http://www.dcomp.sor.ufscar.br/verdi/topicosCloud/nosql_artigo.pdf.

Veen, Jan Sipke van der, Bram van der Waaij, e Robert J. Meijer. 2012. «Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual». Em , 431–38. IEEE. https://doi.org/10.1109/CLOUD.2012.18.

Voorhees, Alex. 2014. «Everything You Need to Know About Google App Engine In Less Than 4 Minutes». 23 de Outubro de 2014.

https://www.cloudbakers.com/blog/everything-you-need-to-know-about-google-app-engine-in-less-than-4-minutes.

Wampler, D., e T. Clark. 2010. «Guest Editors' Introduction: Multiparadigm Programming». *IEEE Software* 27 (5):20–24. https://doi.org/10.1109/MS.2010.119.

Wiese, Lena. 2015. «Polyglot Database Architectures= Polyglot Challenges.» Em *LWA*, 422–426. http://ceur-ws.org/Vol-1458/H05_CRC85_Wiese.pdf.

Worsley, John, e Joshua D. Drake. 2002. *Practical PostgreSQL*. O'Reilly Media, Inc.

Zhao, Gansen, Weichai Huang, Shunlin Liang, e Yong Tang. 2013. «Modeling MongoDB with Relational Model». Em , 115–21. IEEE. https://doi.org/10.1109/EIDWT.2013.25.

Zhao, Gansen, Qiaoying Lin, Libo Li, e Zijing Li. 2014. «Schema Conversion Model of SQL Database to NoSQL». Em , 355–62. IEEE. https://doi.org/10.1109/3PGCIC.2014.137.

Zuber-Skerritt, O. 2003. *New Directions in Action Research*. Taylor & Francis. https://books.google.pt/books?id=moeRAgAAQBAJ.

# ANEXOS

# ANEXO A

## Rule 1

### The Information Rule

All information in the relational database is represented in exactly one and only one way—by values in tables.

## Rule 2

### Guaranteed Access Rule

Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.

## Rule 3

### Systematic Treatment of NULL Values

NULL values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational RDBMS for representing missing information in a systematic way, independent of data type.

## Rule 4

### Dynamic Online Catalog Based on the Relational Model

The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data.

# Rule 5

### Comprehensive Data Sublanguage Rule

A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible: a. data definition b. view definition c. data manipulation (interactive and by program) d. integrity constraints e. authorization f. transaction boundaries (begin, commit, and rollback).

# Rule 6

### View Updating Rule

All views that are theoretically updateable are also updateable by the system.

# Rule 7

### High-Level Insert, Update, and Delete

The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data.

# Rule 8

### Physical Data Independence

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

# Rule 9

**Logical Data Independence**

Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

# Rule 10

**Integrity Independence**

Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

# Rule 11

**Distribution Independence**

The data manipulation sublanguage of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically centralized or distributed.

# Rule 12

**Non-Subversion Rule**

If a relational system has or supports a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

Adaptado de: (E. F. Codd 1985b)