FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO


FEUP

# Unifield - Indexação de informação estruturada para pesquisas não estruturadas

**André Manuel Pereira dos Santos**

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Sérgio Nunes (Doutor)

Proponente: José Bonnet - PT Inovação (Eng.)

21 de Julho de 2011

# Unifield - Indexação de informação estruturada para pesquisas não estruturadas

**André Manuel Pereira dos Santos**

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: Eduarda Mendes Rodrigues (Doutora)

Vogal Externo: Rui Oliveira (Doutor)

Orientador: Sérgio Nunes (Doutor)

---

13 de Julho de 2011

# Resumo

Os sistemas de pesquisa na *web* tornaram popular a pesquisa por palavras-chave. Este método de pesquisa melhorou claramente a usabilidade das pesquisas na *web* face às tradicionais pesquisas estruturadas que conhecemos da área das bases de dados relacionais. Este tipo de pesquisas envolvem um formulário com várias caixas de texto que testam determinados atributos na base de dados. Deste modo, tem havido um esforço crescente nos últimos anos, tanto a nível académico como a nível empresarial, para a criação de aplicações que permitam a pesquisa por palavras-chave sobre bases de dados estruturadas. Por um lado, surgiram extensões nos sistemas de gestão de base de dados relacionais para dar resposta a esta necessidade, e por outro, apareceram vários sistemas open-source baseados em modelos não relacionais. Este trabalho compara as capacidades reais das plataformas existentes para implementar eficientemente a pesquisa por palavras-chave em informação estruturada. Para tal, foram definidos os casos de uso mais relevantes para a PT Inovação, maioritariamente operações de inserção e consulta de informação na base de dados, e sobre estes foi desenhada uma bateria de testes de modo a colocar as plataformas à prova. O objectivo principal é simular e prever os seus comportamentos em ambientes de produção realistas. Os resultados experimentais indicam que ambas as tecnologias têm um desempenho semelhante nas fases de indexação e pesquisa nos testes menos exigentes. À medida que os requisitos a nível de indexação aumentam, o desempenho dos sistemas relacionais degrada-se substancialmente. É essencial adaptar a configuração de um sistema relacional ao cenário onde está a executar, para obter um desempenho mais razoável nos cenário mais exigentes. Por seu lado, os sistemas que seguem um modelo não relacional confirmam a sua vocação para a indexação, ao apresentarem, na sua configuração de base um desempenho que sofre pouca influência pela exigência do teste a nível de indexação, quer por meio da quantidade de informação, quer com o número de atributos a indexar. Os testes efectuados permitem concluir que um único sistema não consegue ter um rendimento muito eficiente em áreas tão distintas como a indexação e o armazenamento, deste modo, a adopção de uma solução mista, através da coexistência de tecnologias relacionais e não relacionais, permitiria explorar os benefícios de ambas.

# Abstract

Web search systems have popularized keyword-based search. This search paradigm clearly improved the usability of web searches when compared to the traditional structured search offered by relational database systems. The interface of the latter search method is based on forms with several text boxes and each one of them will be matched against a specific attribute in the database. In the past few years, both academic and business worlds have devoted a lot of research to the creation of systems that allow efficient keyword-based search over structured databases. The outcome of this effort resulted in several plugins for RDBS and open source systems focused on keyword search based on non-relation models. The work reported in this document compares the actual capabilities of the existing platforms that follow both relational and non-relational models to implement efficiently keyword-based search over structured data. In order to do so, the most relevant use-cases of PT Inovação were gathered - mostly, insert and query operations - and it was designed a benchmarking system to cover those use cases. The main goal of this task consisted in the simulation and observation of the behavior of the platforms in realistic production environments. Experimental results put it clear that less demanding tests indicated that both technologies have a similar performance both at indexing and at searching. Although, as the indexing requirements increase, the performance of the relational systems go down very quickly. So, it is mandatory to adapt the configuration of these systems to the scenario where it is running to obtain a reasonable performance in demanding environments. On the other hand, non-relation systems performed really well at indexing because, on their basic settings, their performances were minimally affected when exposed to more complex indexing scenarios, either by the increase of the amount of information indexed or the number of attributes to index. Finally, the benchmark let us conclude that a single system can't have a very efficient performance in so complex fields, such as indexing and storage. Thus, the adoption of a mixed solution with both non-relational and relational plataforms would allow us to explore the benefits of both plataforms.

# Agradecimentos

Gostaria de expressar aqui o meu agradecimento geral a todos aqueles que de alguma forma contribuíram para o sucesso destre trabalho.

Em particular, estou muito grato ao meu orientador, o professor Sérgio Nunes, pelo apoio e ajuda constantes. Os seus conselhos foram cruciais para guiar o meu trabalho.

Agradeço ainda à PT Inovação e em particular ao Eng. José Bonnet pela oportunidade que me proporcionaram e pelo excelente acolhimento que tive ao longo destes últimos cinco meses. Quero ainda agradecer aos meus colegas da PT Inovação que foram o meu suporte diário. Sou obrigado a destacar o meu colega João Peixoto que foi sempre o meu primeiro amparo. Estou-lhe muito grato pela sua atitude e compreensão, sempre que necessitei de ajuda.

Aos meus colegas de curso, pelo apoio ao longo destes últimos cinco anos.

À minha família que me apoiou e incentivou ao longo do meu percurso académico.

À minha namorada, Joana Quintela, pela inspiração e ajuda nos momentos que mais precisei.

André Santos

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Abreviaturas e Símbolos

| | |
|---|---|
| API | *Application Programming Interface* |
| CPU | Unidade Central de Processamento |
| CRUD | Actualizar, Apagar, Inserir, Pesquisar |
| JIT | *Just-in-time* |
| HTTP | *Hypertext Transfer Protocol* |
| ODBC | *Open DataBase Connectivity* |
| OJDBC | *Oracle Java DataBase Connectivity* |
| RAM | *Random-Access Memory* |
| RC | Rede Candidata |
| RI | Recuperação de Informação |
| SGBD | Sistema de Gestão de Base de Dados |
| XML | *Extensible Markup Language* |
| XPath | *XML Path Language* |

# Capítulo 1

# Introdução

## 1.1 Contexto

Actualmente, a concorrência feroz que atravessa a grande maioria das indústrias tem desencadeado nas organizações processos para repensar as suas posições no mercado, que incluem a estratégia e as abordagens que seguem e os objectivos que traçaram. A área do desenvolvimento de software não é excepção e é notório um foco crescente na criação de produtos e serviços com maior eficiência e com o maior valor possível para o cliente. Com este fim, a produção de software deve seguir uma filosofia centrada no utilizador e não no sistema [Fer08], para que as necessidades do utilizador tomem a devida prioridade desde o início do processo de desenvolvimento de software.

Um dos conceitos que tem ganho cada vez mais relevância, particularmente na produção de software, é a *usabilidade*. Aplicações feitas, por exemplo à escala da Internet, estão expostas a uma vasta audiência. Naturalmente, a interacção com o utilizador deve ser projectada nessa base, especialmente para contemplar utilizadores com diferentes níveis de instrução e incapacidades físicas. No caso da Internet, Jakob Nielsen afirma que "a usabilidade é uma condição necessária de sobrevivência" [Gar07]. Torna-se claro, portanto, a necessidade de tornar os testes de usabilidade uma parte integrante dos processos de desenvolvimento de software.

Nas organizações, os testes de usabilidade têm impacto positivo no suporte prestado ao cliente. Má usabilidade constitui uma das principais razões para a necessidade de prestar suporte técnico aos utilizadores [Cor05]. A nível interno, a incorporação da usabilidade nos processos de produção de software causa a diminuição dos custos de aprendizagem e o aumento da produtividade [Cor05]. Por outro lado, os testes de usabilidade melhoram a aceitação do produto. A aceitação do utilizador frequentemente correlaciona-se com a fidelidade à empresa e ao produto, o que significa que um utilizador motivado por uma boa experiência provavelmente irá recomendar um produto a outros utilizadores [Cor05].

Tomando um caso tão conhecido com o Facebook, para manter a sua incrível comunidade de 600 milhões de utilizadores[1] já tiveram lugar várias remodelações do sistema com o intuito principal de tornar a interface mais simples e limpa [Pat08], o que confirma actualmente a relevância do tema *usabilidade* para as grandes organizações.

Em Portugal, o grupo Portugal Telecom lida diariamente com grandes volumes de informação e os colaboradores e clientes que acedem à informação têm níveis muito diferentes de instrução. Como tal, a Portugal Telecom, para além dos requisitos de eficiência, necessita de sistemas de pesquisa usáveis. É nesta situação que se insere esta dissertação.

No âmbito desta dissertação foi produzido um artigo que posteriormente foi submetido e aceite na Conferência INForum 2011[2].

## 1.2 Motivação e Objectivos

As bases de dados estruturadas são o modelo mais amplamente adoptado para armazenar informação [LWL08], logo a maior parte das pesquisas efectuadas são feitas sobre este modelo de dados. A interface mais usual deste tipo de sistemas consiste num formulário onde existe um conjunto de caixas de texto para preenchermos os critérios de pesquisa. Na Figura 1.1 temos um exemplo de uma interface com as características indicadas.

Temos nos motores de busca mais conhecidos da Internet (Google, Yahoo, Bing) bons exemplos de aplicações que oferecem a pesquisa por palavras-chave (ou pesquisa não estruturada). Com apenas uma caixa de texto, o utilizador ao inserir os termos da pesquisa obtém instantaneamente sugestões de resultados. Este processo, conhecido como pesquisa por palavras-chave, torna as pesquisas mais convenientes ao proporcionar um ambiente bastante mais natural ao utilizador.

Ao comparar a pesquisa estruturada como apresentada na Figura 1.1 com a pesquisa não estruturada, é evidente que esta última proporciona uma interface mais amigável ao utilizador. O mesmo acontece na geração das ocorrências a uma pesquisa. As bases de dados estruturadas suportam o modelo booleano de pesquisa, ou seja, cada pesquisa devolve todos os tuplos que satisfazem as condições da pesquisa [ACDG03]. Há a possibilidade, no entanto, de utilizar mecanismos de ordenação básicos. Por seu lado, a pesquisa por palavras-chave como a que conhecemos dos motores de busca da Internet, incorpora conceitos de Recuperação da Informação (RI) para fornecer mecanismos automáticos de ordenação por relevância dos resultados de uma pesquisa, o que se tem revelado uma abordagem bem sucedida [LYMC06].

Constata-se portanto, que seria bastante interessante aproveitar o conceito de pesquisa por palavras-chave, que nos motores de busca é aplicado a informação não estruturada, para sistemas que armazenam a informação de forma estruturada, como a grande maioria

[1] A barreira dos 600 milhões de utilizadores foi ultrapassada em Dezembro de 2010 [Car11].

[2] `http://inforum.org.pt/INForum2011`

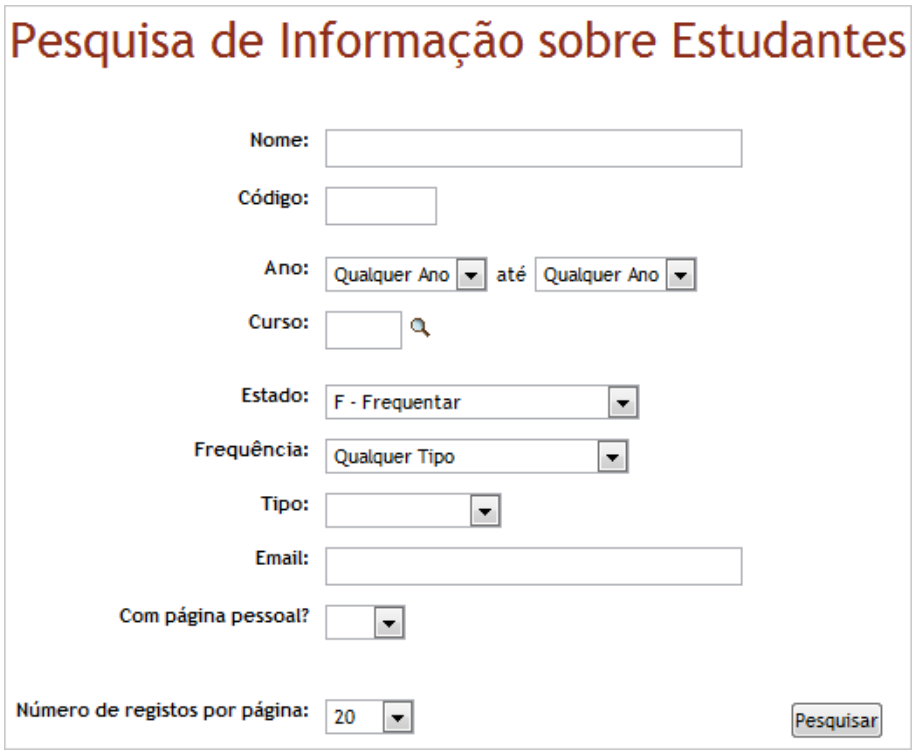



Figura 1.1: Formulário de pesquisa de estudantes no sistema de informação SiFEUP.

dos sistemas operados pela PT Inovação. No entanto, vários problemas se colocam para implementar este tipo de pesquisa em sistemas relacionais.

## 1.3 Estrutura do Documento

O resto do documento está estruturado da seguinte forma: no Capítulo 2 é feita uma revisão do estado da arte e são detalhados métodos e sistemas que permitem aplicar a pesquisa não estruturada sobre base de dados estruturadas, no Capítulo 3 é descrito o problema proposto, o Capítulo 4 enuncia a metodologia e solução utilizada para resolver o problema, no Capítulo 5 são detalhados os resultados das experiências efectuadas e no Capítulo 6 são apresentadas as conclusões do trabalho e perspectivas de trabalho futuro.

# Capítulo 2

# Estudo das Tecnologias

## 2.1 Introdução

Inspirado pelo grande sucesso da pesquisa por palavras-chave com técnicas de Recuperação de Informação na Internet, o mesmo modelo de pesquisa em base de dados estruturadas emergiu como um tópico de investigação que tem tido bastante actividade nos últimos anos [LYMC06]. Para melhor distinguir os dois tipos de pesquisa, vamos acompanhar o fluxo de cada pesquisa em paralelo, quer do ponto de vista do utilizador, quer da perspectiva do sistema.

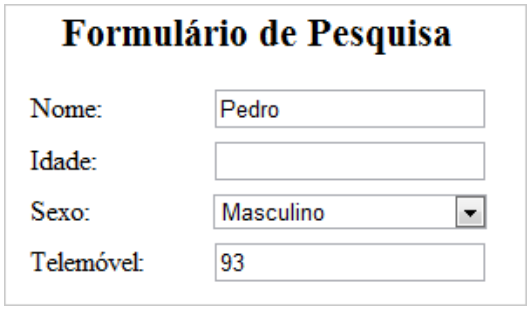



Figura 2.1: Exemplo de formulário de uma pesquisa estruturada.

Nas pesquisas estruturadas são indicadas explicitamente as restrições a aplicar a cada atributo. Na Figura 2.1 temos um exemplo de um formulário onde existe um mapeamento intuitivo entre cada caixa de texto e o atributo na base de dados a que se refere. É bastante perceptível o propósito da pesquisa, ou seja, quais os clientes da tabela Cliente com o nome "Pedro", o sexo "masculino" e que tenham um telemóvel em que o número contenha os dígitos "93".

Quanto às pesquisas não estruturadas, temos uma interface mais simples, amigável e que permite que o utilizador se exprima de uma forma mais natural. Num único campo o utilizador insere todos os termos de pesquisa que pretende. Na Figura 2.2 temos um

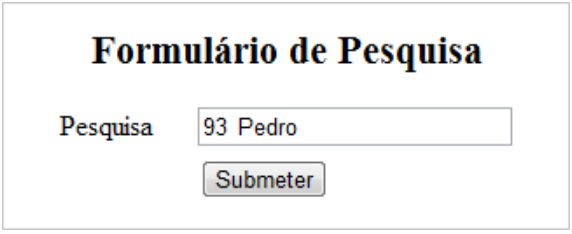



Figura 2.2: Exemplo de formulário de uma pesquisa não estruturada.

exemplo de um formulário com estas características. Atendendo ao critério de pesquisa da figura "93 Pedro", podemos deduzir que se pretende o conjunto dos clientes que têm o nome "Pedro" e que tenham um telemóvel em que o número contenha os dígitos "93". No entanto, é inerente a subjectividade dos critérios da pesquisa. Os dígitos "93" podem significar um critério para restringir o número da porta, ou outros dados pessoais como o número de contribuinte, o ano de nascimento ou até a idade de um cliente.

Voltando ao formulário da Figura 2.1, os critérios de pesquisa que este contém podem ser traduzidos directamente numa consulta SQL como a que se segue:

```
Select *
From Cliente
Where sexo = "M"
And nome like "Pedro"
And telemovel like "93"
```

A nível de implementação, a pesquisa não estruturada tem uma maior complexidade. Devido à subjectividade já referida, não há a possibilidade de converter automaticamente o formulário de pesquisa numa consulta SQL. A recente investigação na área opta invariavelmente pela construção de um sistema com base em técnicas de Recuperação de Informação [HGP03, LYMC06] para implementar este modelo de pesquisa em base de dados estruturadas.

Genericamente, todos os sistemas estudados nesta área envolvem um conjunto semelhante de etapas. A Figura 2.3 retrata uma arquitectura modelo. Em primeiro lugar, são precisos mecanismos de indexação para cobrir a necessidade de em cada consulta obter as referências na base de dados de cada palavra-chave. Após esta etapa devem ser construídos os resultados candidatos à consulta e por fim, devem ser ordenados e apresentados ao utilizador. Um conceito que atravessa todo o sistema e que é abordado ao longo deste capítulo é a questão da granularidade da informação.

Para mais facilmente transmitir os conceitos que vão ser abordados nas próximas secções serão feitas analogias a um único exemplo de uma empresa de telecomunicações:

*Uma empresa de telecomunicações actualmente armazena a informação em*

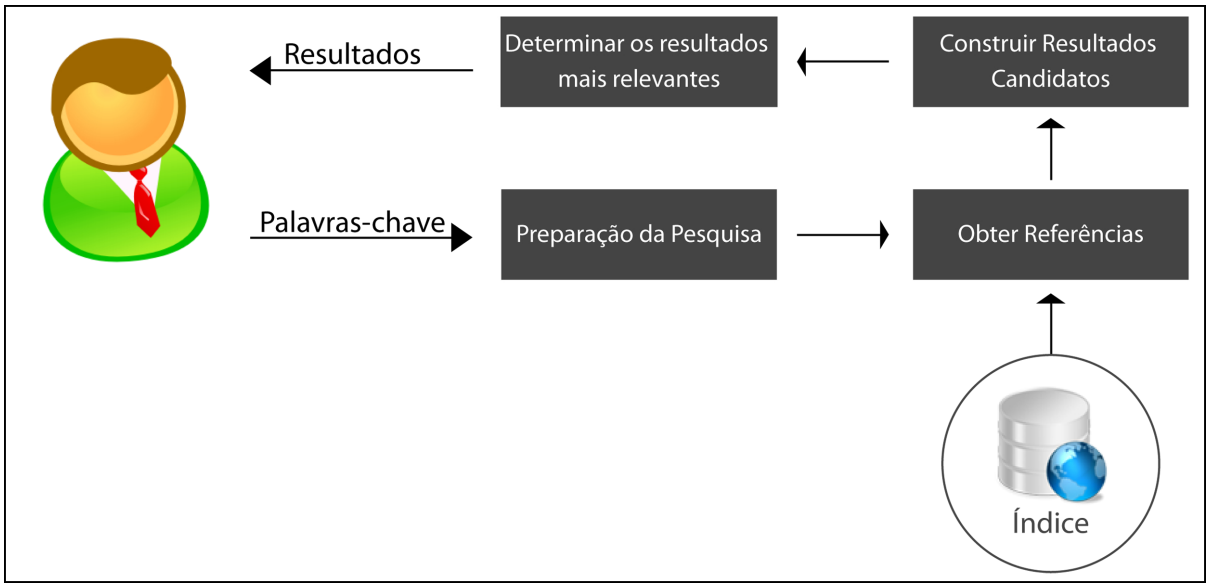



Figura 2.3: Arquitectura genérica de um sistema que permite a pesquisa por palavras-chave.

*bases de dados relacionais. No entanto, uma grande parte dos seus colaboradores, nomeadamente nos seus call centers, gasta bastante tempo a pesquisar as fontes de informação da empresa no atendimento ao cliente. Recentemente a empresa decidiu alterar os formulários de pesquisa, para uma interface mais amigável com apenas um campo de pesquisa. O sistema de informação tem um vasto conjunto de relações: Cliente, Tarifário, Equipamento, Registo de chamadas e Saldo, entre outras, que armazenam uma grande quantidade de informação.*

Seguidamente são apresentadas quatro tabelas com um pequeno conjunto de tuplos das relações Cliente (Tabela 2.1), Equipamento (Tabela 2.2), Equipamento-Cliente (Tabela 2.3) e Chamada (Tabela 2.4).

Tabela 2.1: Tabela Cliente.

| **ID-Cliente** | **Nome** | **Idade** |
|---|---|---|
| c1 | João | 40 |
| c2 | Rui | 22 |
| c3 | Pedro | 35 |

## 2.2 Indexação

Para cada pesquisa à base de dados, formula-se o seguinte problema:

Tabela 2.2: Tabela Equipamento.

| ID-Equipamento | Marca | Modelo | Especificações |
|---|---|---|---|
| e1 | Nokia | X6 | 5MP, GPS, 3.2", ... |
| e2 | Apple | iPhone4 | 5MP, GPS, 3.5", ... |
| e3 | Samsung | Galaxy S | 5MP, GPS, 4", ... |
| e4 | Nokia | 2730 | 2MP, MP3, ... |
| e5 | Samsung | Apollo | 3MP, GPS, MP3, ... |

Tabela 2.3: Tabela Equipamento-Cliente.

| ID-Equipamento-Cliente | ID-Cliente | ID-Equipamento |
|---|---|---|
| ec1 | c1 | e3 |
| ec2 | c2 | e1 |
| ec3 | c2 | e2 |

Tabela 2.4: Tabela Chamada.

| ID-Chamada | ID-Cliente | ID-Equipamento | Data | Duracao |
|---|---|---|---|---|
| ch1 | c1 | e3 | 29-01-2011 | 00:00:45 |
| ch2 | c2 | e2 | 31-01-2011 | 00:00:25 |
| ch3 | c2 | e1 | 01-02-2011 | 00:05:10 |
| ch4 | c2 | e2 | 02-02-2011 | 00:03:05 |
| ch5 | c1 | e3 | 02-02-2011 | 00:01:20 |

> Dado o conjunto de palavras-chave de uma consulta, como obter eficientemente a informação referenciada na base de dados?

Não é razoável pesquisar um conjunto de palavras-chave numa base de dados com alguns milhões de registos e esperar obter um tempo de resposta aceitável. A solução para este problema passa pela criação de índices sobre os atributos que se pretende pesquisar. No entanto, existem diversas situações onde não é benéfico criar um índice, uma vez que este tem custos associados [ZM06]. Nomeadamente, se não for previsto uma quantidade significativa de consultas, se a informação for muito volátil ou se houver pouca informação para pesquisar deve-se optar por não utilizar mecanismos de indexação. Para este trabalho, os ambientes de produção com que a PT Inovação lida diariamente obrigam a que a informação seja indexada. Nos seus sistemas mais exigentes diariamente são inseridos alguns GBytes de informação e por norma, estes respondem a várias dezenas de consultas por segundo.

A maior parte dos índices criados em bases de dados são muito grandes para caber em memória, assim, têm de estar armazenados no disco. Actualmente as operações de leitura e escrita no disco são os processos mais custosos em cada transacção de uma base de dados. Assim, o maior objectivo dos índices armazenados no disco é minimizar as operações de leitura e escrita no disco [LNT00b]. Por outro lado, com o advento da tecnologia do suporte físico, os sistemas começam a estar equipados com grandes

quantidades de memória primária, o que permite em alguns casos que o índice resida totalmente na memória principal. Neste caso e sem o entrave do tempo consumido com acessos ao disco, o objectivo principal é minimizar o tempo de computação, evitando também atingir o limite de memória utilizada [LC86].

Existem muitas implementações para os índices das bases de dados. Como tal, vão ser abordadas as mais relevantes. Uma das implementações mais utilizadas para um índice é a B-Tree [LNT00b]. Este tipo de índice é uma generalização das árvores binárias, na medida em que permite ter mais do que dois filhos por nó. Por norma, é o índice adoptado por omissão nas bases de dados relacionais (Oracle, MySQL, Microsoft SQL Server, entre outros) [LNT00b]. Ao utilizar um índice deste tipo evita-se a ordenação de grandes quantidades de dados, uma vez que a estrutura do índice mantém sempre a informação ordenada. Na Figura 2.4 apresenta-se um fragmento de um índice B-Tree. A informação é mantida em forma de árvore e para pesquisar um determinado valor percorre-se a árvore começando na raiz. Na figura está descrita uma árvore de ordem 3 (número máximo de filhos por cada nó) e de profundidade 3 (número de níveis a contar a partir da raiz). Para cada nó é feito um teste e o seu resultado determina o próximo ramo a seguir e a assim sucessivamente, até se atingir uma folha da árvore. Imaginemos que se pretendia encontrar o valor "150" na árvore representada na Figura 2.4. Como "150" é superior ao valor da raiz, é escolhido o ramo da direita. No nível 2, como "150" encontra-se entre os dois valores do nó, então o caminho continuava pelo ramo central. Depois na folha, percorrem-se os seus valores até se encontrar o pretendido. Este índice garante uma ordem de complexidade logarítmica nas operações de inserir, pesquisar e apagar [LNT00b].

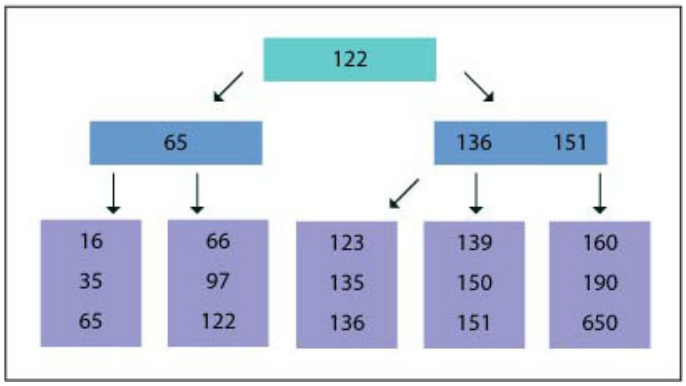



Figura 2.4: Um fragmento da estrutura do índice B-Tree.

Uma das situações onde o índice B-Tree não é vantajoso ocorre quando a cardinalidade da coluna do índice, conjunto de valores distintos, é muito reduzida. Deste modo, não se obtém a selectividade pretendida e o número de acessos ao disco aumenta consideravelmente. Para resolver este problema foi concebido um índice que se torna vantajoso quando os atributos têm baixa cardinalidade: o índice Bitmap [CI98]. A sua representação

está demonstrada na Tabela 2.5. Ao definir um índice Bitmap sobre o atributo "Estado" é criado um vector de bits por cada valor diferente que este atributo possa ter. O tamanho dos vectores corresponde ao número de linhas da relação onde se encontra. Cada posição do vector toma o valor 1, quando o valor do atributo "Estado" da linha correspondente é igual ao valor a que o vector se refere. No caso do vector associado ao valor "Ocupado", como nos registos com identificadores 2, 4 e 5 o valor do atributo "Estado" é "Ocupado" então o valor correspondente no vector é 1 e nos casos contrários é 0.

O índice Bitmap alcança bons resultados a nível de espaço e de tempo, comparativamente a outros índices, quando aplicado sobre colunas com baixa cardinalidade. Utilizando operações bit-a-bit sobre os vectores permite encontrar de um modo muito eficiente os registos que validam uma determinada condição [CI98]. Quanto maior for a cardinalidade do atributo, maior é o conjunto de vectores necessário, e consequentemente, a dimensão espacial do Bitmap aumenta exponencialmente [CI98]. A nível de tempo, com o aumento da cardinalidade a performance piora em proporção. Uma outra situação desfavorável ao índice Bitmap ocorre quando são frequentes as operações de actualizar e apagar, visto que o índice incorre em operações muito custosas para actualizar os vectores [CI98]. Como exemplo, na Tabela 2.5, se fosse apagado o registo com identificador 3, era necessário deslocar um nível para cima todas as posições de cada vector desde o registo com identificador 4 até ao fim da tabela.

Tabela 2.5: Representação de um índice Bitmap sobre um atributo.

| **ID** | **Estado** | | **Bitmap** | |
|---|---|---|---|---|
| | | **Livre** | **Ocupado** | **Em espera** |
| 1 | Livre | 1 | 0 | 0 |
| 2 | Ocupado | 0 | 1 | 0 |
| 3 | Livre | 1 | 0 | 0 |
| 4 | Ocupado | 0 | 1 | 0 |
| 5 | Ocupado | 0 | 1 | 0 |
| 6 | Em espera | 0 | 0 | 1 |

Outra implementação para um índice muito comum nos SGBD é o índice *Hash* [LC86]. Neste índice é utilizada uma função de *hashing* que calcula para cada valor de entrada uma chave que é útil para saber onde posicionar cada valor na tabela de *Hash*. Este processo está representado na Figura 2.5. Idealmente cada valor mapeia uma única chave, o que resulta em tempos de inserção e pesquisa constantes. No entanto, na prática isto nunca acontece [LC86]. Assim, é necessário desenvolver mecanismos para lidar com as colisões, que ocorrem quando uma mesma chave corresponde a vários valores. Para resolver a colisão da Figura 2.5, "João" ficou com o lugar livre imediatamente a seguir à chave de "Clara". Existem vários tipos de resolução de colisões, entre os quais se destaca *Separate Chaining*, onde cada posição da tabela de *Hash* aponta para uma lista de valores que mapeados por uma determinada chave [LC86]. Para que o índice *Hash* seja vantajoso, não

devem ocorrer muitas colisões, o que está dependente da função de *hashing* e do tamanho da tabela. Se a função de *hashing* não conseguir dispersar bem os valores pela tabela de *Hash*, o que pode ser conseguido com conhecimento prévio do tipo de informação que vai armazenar, este índice não se revela uma boa opção [LC86].

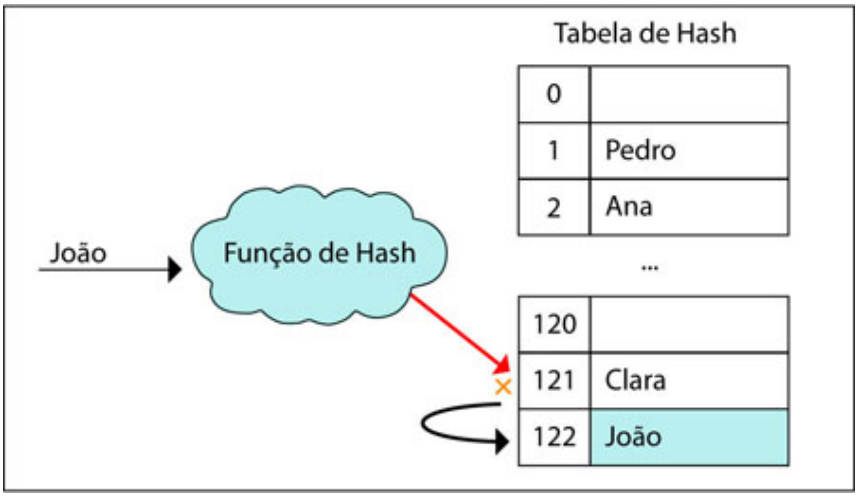



Figura 2.5: Representação do processo de inserção de um valor numa tabela de *Hash*.

Foram já cobertos os índices B-Tree, Bitmap e *Hash*. As implementações originais destes índices não são eficientes na pesquisa parcial de um valor. Nos sistemas relacionais, quando é feita uma pesquisa de uma palavra num campo de texto, em linguagem SQL utilizando a cláusula LIKE, independentemente de o campo ter ou não um índice é sempre feita uma travessia em toda a tabela (*full table scan*), o que é impraticável em sistemas com grande dimensão [Bur].

Para tornar eficiente a pesquisa por palavras-chave surgiram os índices invertidos. Um índice invertido consiste numa lista ordenada de palavras-chave, onde cada uma mapeia um conjunto de referências para a base de dados [HFBYL92]. Na sua representação básica, um índice invertido é constituído por um vocabulário, conjunto distinto de palavras-chave, onde para cada palavra é guardada a contagem dos documentos que a contêm e um apontador para o início de uma lista invertida. Cada lista invertida armazena um conjunto de referências para os tuplos da base de dados juntamente com a frequência da palavra-chave correspondente em cada tuplo [HFBYL92]. Esta representação pode ser consultada na Tabela 2.6, onde é apresentado um fragmento de um índice invertido referente ao exemplo descrito no início deste capítulo.

Este tipo de índice é suportado por vários SGBD. Nesta categoria destaca-se o índice CONTEXT do módulo Oracle TEXT [Cor09b], os índices *fulltext* do MySQL [Cor] e do Microsoft SQL Server [Lop08]. Ao operar em bases de dados estruturadas, com relações entre tuplos de diferentes tabelas, os índices invertidos destes sistemas não capturam a semântica na fase de indexação. O mesmo é dizer que referenciam registos de

Tabela 2.6: Fragmento de ficheiro invertido.

| **Palavra-chave** | **Frequência** | **Lista Invertida** |
|---|---|---|
| 5MP | 3 | ( <e1 , 1> <e2 , 1> <e3 , 1> ) |
| Apple | 1 | ( <e2 , 1> ) |
| GPS | 3 | ( <e1 , 1> <e2 , 1> <e3 , 1> ) |
| Iphone | 1 | ( <e2 , 1> ) |
| ... | ... | ... |
| Samsung | 1 | ( <e3 , 1> ) |
| ... | ... | ... |

tabelas sem tomar em consideração as eventuais relações que possam ter com outros registos [LYMC06].

A origem do índice invertido advém da área da Recuperação de Informação, onde a unidade lógica de pesquisa é genericamente o documento [LYMC06]. Nesta área, os documentos são mapeados no índice e as ocorrências básicas a uma pesquisa são documentos. Uma vez que um documento é concebido para agregar toda a informação que lhe diz respeito, não se coloca a questão da perda de semântica. Devido à normalização das bases de dados relacionais, as unidades lógicas de informação podem estar fragmentadas ao longo de várias tabelas. Dado um conjunto de palavras-chave, um potencial resultado a uma pesquisa combina tuplos de várias tabelas. Assim, para não perder a semântica em bases de dados relacionais é necessário adicionar uma camada adicional para agregar a informação que esteja relacionada entre si (registos ligados por chaves estrangeiras). Esta questão será tratada em detalhe na Secção 2.3.

Para além dos SGBD existem sistemas open-source construídos especificamente para permitir a pesquisa por palavras-chave sobre informação estruturada tais como o Sphinx Search e o Apache Lucene que utilizam índices invertidos. Estes dois sistemas serão abordados em mais detalhe na Secção 2.5. A revisão bibliográfica permite concluir que foram criados alguns sistemas, maioritariamente a nível académico, que capturam a semântica na fase de indexação. Seguidamente serão discutidas as estruturas que utilizam.

Intuitivamente, tal como apresentado na Tabela 2.6, podemos pensar no tuplo como a referência mais adequada em bases de dados relacionais. No entanto, os resultados de uma pesquisa, como já foi referido, são construídos por norma pela combinação de vários tuplos. Assim, podemos aproveitar para estabelecer uma analogia com Recuperação de Informação de modo a adoptar o conceito de *documento virtual* [SW05, WLK03] que pode servir mutuamente para efeitos de indexação e de apresentação de resultados. Este conceito será explorado mais à frente.

No caso da granularidade ao nível do tuplo para efeitos de indexação, que é suportada pelos SGBD e pelos sistemas DISCOVER [HP02], ITREKS [ZW07] e BANKS [BHNC02], ao fornecer como entrada cada palavra-chave ao índice, são obtidos os tuplos de cada relação que a contém. Esta abordagem pode ser implementada com recurso

às funcionalidades do SGBD, como é feita por exemplo no sistema DISCOVER onde são utilizados os índices de texto do SGBD para indexar cada relação e depois é implementado externamente um meta-índice para agregar as referências de cada índice. Para melhorar significativamente a eficiência desta etapa, idealmente existiria apenas um único índice que mapeava as palavras-chave nos tuplos de todas as relações [HP02]. Contudo, nesta situação seria necessário desenvolver um índice sem o suporte do SGBD.

É possível também recorrer a um método de indexação com granularidade a nível do documento virtual, como já foi referido neste capítulo. Este conceito é suportado pelos sistemas EKSO [SW05] e DbSurfer [WLK03]. Um documento virtual pode ser pensado como um registo de uma base de dados não normalizada, como se exemplifica na Tabela 2.7. Neste exemplo, cada registo da tabela reúne toda a informação de um cliente.

Os documentos virtuais podem ser criados em duas etapas. Em primeiro lugar, criam-se *objectos de texto* que podem ser descritos como a união de tuplos através de uma sequência de uma ou mais junções por chave estrangeira. Tomando como exemplo a base de dados na Tabela 2.7, é feita a união dos tuplos das relações Cliente (Tabela 2.1), Equipamento (Tabela 2.2) e Chamada (Tabela 2.4), pelas chaves estrangeiras *ID-Cliente* e *ID-Equipamento*. Depois segue-se o passo de concatenação dos atributos dos objectos de texto que origina os documentos virtuais.

Tabela 2.7: Base de dados não normalizada.

| **Nome** | **Idade** | **Marca** | **Modelo** | **Especificações** | **Data** | **Duração** |
|---|---|---|---|---|---|---|
| João | 40 | Samsung | Galaxy S | 5MP, GPS, 4", ... | 29-01-2011 | 00:00:45 |
| | | | | | 02-02-2011 | 00:01:20 |
| Rui | 22 | Nokia | X6 | 5MP, GPS, 3.2", ... | 01-02-2011 | 00:05:10 |
| | | Apple | Iphone4 | 5MP, GPS, 3.5", ... | 31-01-2011 | 00:00:25 |
| | | | | | 02-02-2011 | 00:03:05 |
| Pedro | 35 | | | | | |

Para evitar problemas de sincronização com a base de dados, a indexação dos documentos virtuais pode ser feita aproveitando os índices de texto fornecidos dos SGBD [SW05]. A indexação de documentos virtuais requer o armazenamento dos objectos de texto, o mapeamento entre os tuplos de cada relação e os objectos de texto e o mapeamento entre os objectos de texto e os documentos virtuais, de modo a que sempre que haja alterações à informação na base de dados (via inserção, actualização ou remoção), seja viável actualizar os documentos virtuais e assim, manter o índice actualizado e consistente.

## 2.3 Construção de resultados

Para cada pesquisa, após obter as referências na base de dados para cada palavra-chave, coloca-se a seguinte questão:

Que resultados apresentar? Qual a estrutura modelo de um resultado de uma pesquisa?

Novamente temos a granularidade como uma questão muito pertinente. Não existe nenhuma unidade lógica que se revele a estrutura ideal para um resultado a uma pesquisa, como vamos poder perceber nesta secção. Os sistemas estudados e apresentados ao longo deste capítulo foram concebidos com abordagens genéricas. Obviamente que se for possível, a construção de um sistema à medida, ajustado ao domínio do problema, é sempre uma solução com mais capacidade que um sistema com uma abordagem genérica. Os projectos open-source discutidos na Secção 2.5 ultrapassam este obstáculo, ao permitirem que o utilizador defina a estrutura que pretende para as ocorrências de uma pesquisa. Relativamente ao exemplo da empresa de telecomunicações, poder-se-ia definir uma estrutura Cliente que consiste na agregação dos tuplos associados das Tabelas Cliente (Tabela 2.1), Equipamento (Tabela 2.2), Equipamento-Cliente (Tabela 2.3) e Chamada (Tabela 2.4)) que serviria de resposta sempre que fosse pesquisada informação sobre um cliente. Um sistema open-source como o Apache Solr permite definir vários modelos [Fou06] que são interpretados como as estruturas a retornar nos resultados de cada pesquisa

A granularidade a nível do documento virtual pode simultaneamente ser útil para efeitos de indexação e de resultado a uma pesquisa [SW05]. Para cada consulta, após obter os documentos virtuais nos índices, devem ser seleccionados e refinados os resultados candidatos. Estes podem consistir nos documentos virtuais condicionados a uma restrição, por exemplo, conter um número mínimo de palavras-chave da consulta. Inclusivamente, aos documentos virtuais obtidos no indexador podem ser aplicados modelos (*templates*) XML que capturam a estrutura da base de dados e utilizem marcas (*tags*) para codificar atributos e nomes das relações [SW05]. Desta forma os documentos virtuais seriam enriquecidos e seria possível navegar no seu conteúdo por consultas através de linguagens baseadas em XML como XPath[1].

Atentemos à Tabela 2.7, em que cada registo da relação simboliza um documento virtual. Cada registo tem informação de três áreas: Cliente, Telemóvel e Registo de Chamada. No entanto, na perspectiva do utilizador do sistema sempre que se consulta um cliente, independentemente do critério da pesquisa, obter sempre toda a informação do cliente como resultado pode ser bastante penoso, tendo em consideração que um cliente potencialmente tem associado a si um conjunto enorme de registos de chamadas, entre outros dados. Assim, os documentos virtuais devem ser preparados para serem apresentados como resultados, tendo por base o esquema da base de dados e possivelmente informação sobre o domínio do problema, configurada pelo utilizador. Neste caso específico do exemplo da empresa de telecomunicações, a informação em bruto do registo de chamadas

[1] *http://www.w3.org/TR/xpath20/*

pode ficar arredada do resultado final de uma consulta, para não incapacitar o utilizador da extracção da informação pertinente das ocorrências da pesquisa.

Quanto à granularidade a nível do tuplo na fase de indexação como apresentada na Secção 2.2, uma vez que a resposta a uma consulta não pode estar limitada a tuplos individuais, deve contemplar a possibilidade da junção de vários tuplos de várias relações [LYMC06]. Nesta etapa, juntamente com os tuplos que contêm palavras-chave deve ser considerado o esquema das ligações entre as relações existentes na base de dados [ACD02, HP02, BHNC02] para ser viável a construção dos resultados candidatos.

Hristidis et al. [HP02] introduziram o conceito revolucionário de *Rede Candidata* (RC) que representa uma árvore de tuplos que estejam ligados entre si, através da relação de chave estrangeira e que conjuntamente contenham todas as palavras-chave. Este conceito é aproveitado nos sistemas DISCOVER, DBXplorer, BANKS e SPARK [LWL08]. A geração de redes candidatas é naturalmente mais rápida que a geração de documentos virtuais, mas estes últimos são gerados *offline* (apenas para construir o índice) o que não contribui para o tempo gasto a responder a cada consulta. Na Figura 2.6, temos o esquema da base de dados exemplo no lado esquerdo, com as relações ligadas pela chave estrangeira, e no lado direito duas redes candidatas, uma para as palavras-chave "Joao 02-02" e outra para a palavra-chave "Rui".

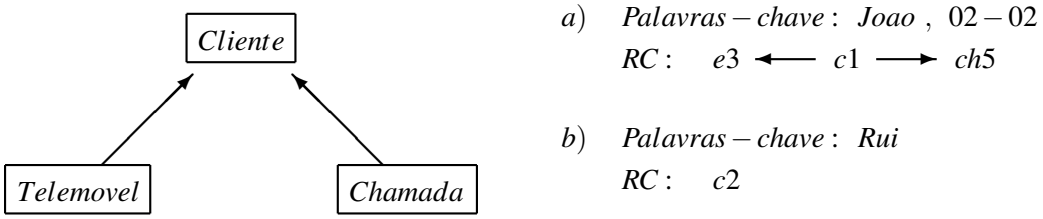



Figura 2.6: Grafo do esquema de base de dados e exemplos de redes candidatas.

O exemplo da rede candidata b) constituída apenas pelo tuplo *c2* demonstra que este conceito necessita de ser ajustado consoante o domínio em que o sistema se enquadra. Regressando ao exemplo do *call center*, sempre que um colaborador inserir apenas o nome de um cliente, no exemplo b) "Rui", este obviamente pretende mais informação do que aquela presente na relação Cliente. Nomeadamente temos no tarifário e no saldo dados indispensáveis sobre um cliente. Assim, tal como acontece com o documento virtual, a estrutura de dados deve ser adaptada ao domínio do problema para produzir resultados úteis na perspectiva do utilizador do sistema.

A geração de redes candidatas deve ser um processo bastante optimizado, uma vez que a maior parte do tempo gasto a responder a uma consulta, nos sistemas que adoptam

a granularidade a nível do tuplo na fase de indexação, é investida nesta etapa [LWL08]. Hristidis et al. [HGP03] referem que o número de uniões de tuplos em cada RC deve ser limitado porque o seu método de criação leva a um número de RC que é exponencial no número de palavras-chave fornecidas. Algoritmos e optimizações úteis para o processo de construção de redes candidatas serão exploradas na Secção 2.4.

Uma consideração importante sobre as redes candidatas é a semântica a utilizar na consulta. Ou seja, se for pretendido que cada resultado contenha todas as palavras-chave é utilizada a semântica AND [BHNC02] e caso cada resultado esteja apenas restringido a ter no mínimo uma palavra-chave, então adopta-se a semântica OR. Torna-se claro que o espaço de resultados é bastante mais restringido quando é utilizada a semântica AND. Por exemplo, numa consulta com K palavras-chave a semântica OR deve considerar todas as redes candidatas que contenham 1, 2, ..., K palavras-chave, já a semântica AND considera apenas redes candidatas com K palavras-chave. Deste modo, a semântica AND é claramente a mais utilizada nos sistemas que adoptam as redes candidatas porque gera um conjunto menor de redes candidatas e como consequência, garante um tempo de resposta mais curto e a capacidade de gerar redes candidatas maiores (redes candidatas com mais uniões entre tuplos) [HGP03].

## 2.4 Ordenação de resultados

Após a fase de selecção dos candidatos a uma pesquisa, deve ser feita a sua ordenação. Esta etapa é motivada por várias razões. O principal motivo é o facto de nem todos os resultados serem igualmente relevantes para o utilizador, nem este ter capacidade para analisar todos os resultados. Logo, estes devem ser ordenados para que o utilizador se possa focar nos mais importantes [LYMC06]. De notar que um dos motivos principais do tremendo sucesso dos motores de busca é a sua eficácia, no sentido de que conseguem efectivamente ordenar as dez páginas *web* mais relevantes a uma consulta em milhões de ocorrências [LYMC06]. Por outro lado, a geração de apenas os K resultados mais relevantes permite tornar a pesquisa mais rápida [HGP03].

A ordenação dos resultados emprega, à imagem do que foi apresentado nas Secções 2.2 e 2.3, técnicas de Recuperação de Informação. Em RI, a etapa de ordenação passa por calcular a pontuação de cada documento com base em métricas relativas ao documento e ao conjunto de todos os documentos do sistema. Já em base de dados estruturadas, para ter em conta a semântica da estrutura da base de dados, as métricas para pontuar os resultados candidatos podem ser relativas ao tuplo, ao atributo, à tabela e/ou à base de dados toda, o que torna o mecanismo de ordenação mais complexo [LYMC06].

Se a granularidade escolhida para os resultados for o documento virtual, intuitivamente teríamos uma aplicação directa das técnicas de ordenação de RI. No entanto, esta solução é pouco eficaz, visto que ao calcular a pontuação de cada documento virtual se

perderia a semântica da estrutura da base de dados [LYMC06]. O mesmo é dizer que não se considera que a informação estava previamente organizada em tabelas com relações entre si, o que tem significado para o utilizador. Dos sistemas que utilizam este nível de granularidade para os resultados, EKSO e DbSurfer, o primeiro não tem mecanismos de ordenação [SW05] e o segundo utiliza uma vertente do algoritmo PageRank [Goo], que originalmente determina uma medida de importância de cada página *web*. De um modo superficial, o DbSurfer cria um grafo em que os nós são os documentos virtuais, representados por páginas *web*, e a as arestas são hiperligações entre as páginas *web*, que na base de dados são representadas pelas relações da chave estrangeira. Este grafo é a entrada do algoritmo PageRank para obter as páginas *web* mais relevantes.

Caso seja pretendido que a granularidade escolhida para os resultados seja a rede candidata, então, como referido na Secção 2.3, é necessário ter um processo bastante optimizado para gerar as redes candidatas. Os sistemas estudados com esta granularidade (DISCOVER, DBXplorer, BANKS e Spark) procuram encontrar as subárvores de tuplos, ou as redes candidatas mínimas, tais que nenhum nó nem nenhuma aresta possa ser removida sem se perder conectividade ou perder ocorrências de palavras-chave na árvore [CWLL09]. Como este problema que se designa pela pesquisa das árvores Steiner é considerado NP-completo [CWLL09], os sistemas referidos definem variações deste problema tais como a limitação do número de palavras-chave de cada pesquisa ou a limitação do número de uniões de tuplos nas redes candidatas, para alcançar tempos de resposta satisfatórios.

Para ordenar as redes candidatas deve ser calculada a pontuação por cada rede candidata. Vários critérios para ordenação já foram propostos, quer com base nas propriedades de cada nó (tuplo) da rede, quer com base nas propriedades da topologia da rede candidata ou então na carga da base de dados. Os métodos mais utilizados em pontuam cada RC com base num misto destes critérios, que vão ser detalhados nos próximos parágrafos [CWLL09].

Para aproveitar as propriedades de cada tuplo para pontuar cada rede candidata, pode ser aplicada uma generalização do famoso algoritmo TF-IDF [LYMC06]: dado um conjunto de documentos e uma consulta (conjunto de palavras-chave), o objectivo é obter os K documentos mais relevantes ou mais similares à consulta. A similaridade entre um documento e uma consulta é formalizada da seguinte forma: dado um vocabulário de **m** palavras-chave, um documento é representado como um vector m-dimensional onde o componente **i** é a frequência da ocorrência (também conhecido como TF - *Term Frequency*) da palavra-chave **i** do vocabulário no documento [ACDG03]. Como a consulta em si é um conjunto de palavras-chave, tem também uma representação vectorial. A similaridade entre um documento e uma consulta é definida pelo produto escalar dos dois vectores correspondentes. Posteriormente, cada componente pode ser escalado com a sua frequência inversa do documento (IDF - *Inverse Document Frequency*) [ACDG03]. O fac-

tor IDF relaciona o número de documentos em que um termo ocorre numa colecção. Este coeficiente é utilizado para sugerir que palavras-chave que ocorrem frequentemente transmitem menos informação do que palavras-chave que ocorrem raramente e deste modo, devem ter um peso inferior.

Para adaptar o algoritmo TF-IDF às base de dados estruturadas existem várias abordagens. Hristidis et al. [HGP03] efectuam uma conversão muito simples: cada valor de cada tuplo é um documento, cada rede candidata é um super documento e cada atributo de cada tabela é uma colecção. Os pesos de cada palavra-chave em cada documento são construídos da forma como nativamente são sugeridos no algoritmo TF-IDF. Quanto ao super documento, cada peso de cada palavra-chave é a média dos pesos da mesma palavra-chave em cada documento. Liu et al. [LYMC06] aproveitam os mesmos conceitos e adicionam um conjunto de normalizações que afectam substancialmente a eficácia da ordenação, nomeadamente a normalização a nível do tamanho da RC (número de uniões entre tuplos) para valorizar redes maiores que intuitivamente contêm mais palavras-chave.

### 2.4.1 Outras ideias para a ordenação

Para além das abordagens de ordenação já apresentadas, que estão intimamente dependentes da granularidade escolhida para o resultado, podem ser ainda exploradas outras ideias para que o mecanismo de ordenação seja mais robusto. Obviamente, o contexto onde o sistema está inserido é fulcral para determinar quais as técnicas ou factores de ordenação mais favoráveis.

A análise da carga da base de dados constitui um critério com muito potencial para ordenar os resultados. A carga da base de dados compreende um conjunto de transacções que podem ser úteis para dar mais ou menos relevo a cada tuplo da base de dados. Mais especificamente, tuplos inseridos ou actualizados recentemente devem ter mais prioridade, para que seja dada mais importância a tuplos mais dinâmicos ou com mais actividade recente. Por outro lado, podemos supor que a frequência de cada tuplo no histórico de consultas à base de dados está directamente relacionada com a sua importância [ACDG03]. Ou seja, quantas mais vezes um tuplo é pesquisado, mais relevância deve ter do que outros menos pesquisados. Como exemplo, vamos supor que no exemplo da empresa de telecomunicações o cliente "Joao" costuma reclamar com muita regularidade. Assim, sempre que a palavra-chave "Joao" fizer parte de uma consulta, faz sentido assumirmos que há uma grande probabilidade de ser novamente o cliente que costuma reclamar com frequência, logo deve ter prioridade em relação a outros clientes com o mesmo nome.

O histórico de pesquisas oferece ainda mais possibilidades para melhorar a eficácia da pesquisa. Imaginemos que um cliente da empresa de telecomunicações apresentada no início do capítulo pretende adquirir um telemóvel. O cliente define um determinado orçamento e um conjunto de funcionalidades que pretende no dispositivo e depois utiliza

o sítio *web* da empresa para encontrar um telemóvel que vá de encontro ao que deseja. Idealmente caso o cliente pesquisasse por exemplo "iPhone4" seria interessante que o sistema incorporasse em possíveis resultados outros dispositivos que tenham características semelhantes. Agrawal et al. [ACDG03] sugerem que o sistema analise o histórico de consultas à base de dados para encontrar tuplos que repetidamente sejam pesquisados em sucessão. Com este método, o cliente receberia como resposta telemóveis que normalmente são pesquisados em conjunto com o telemóvel "iPhone4".

Outras técnicas de RI podem ainda ser exploradas. O conceito *word stemming*, redução de todas as palavras de uma mesma família à palavra-mãe, tem a sua utilidade bastante dependente do domínio onde o sistema está inserido [HGP03]. Pode ser bastante útil, por exemplo, para pesquisar verbos que estão conjugados, no entanto, para o exemplo da empresa de telecomunicações não se percebe nenhuma aplicação prática imediata. Outro conceito, eliminação de *stop words*, trata de anular palavras-chave que ocorram muito frequentemente. Este último tem uma aplicação mais ampla e pode melhorar o mecanismo de ordenação [HGP03].

Em suma, nesta subsecção temos um grande conjunto de critérios com potencial para ordenar resultados. Em geral, as abordagens de ordenação mais adoptadas calculam uma média ponderada de um conjunto amplo de critérios [ACDG03].

## 2.5 Sistemas Open-Source

Nenhum dos sistemas apresentados antes desta secção disponibiliza a sua implementação. Caso se pretenda construir um sistema para implementar a pesquisa não estruturada em base de dados estruturadas onde a eficiência seja o requisito principal há eventualmente a necessidade de construir um sistema personalizado com um misto de conceitos e componentes dos vários sistemas estudados. Contudo, existe uma grande quantidade de projectos open-source nesta área que prometem ser uma excelente solução.

Existe um grande conjunto de sistemas de pesquisa open-source: ASPSeek, BBDBot, Eureka, Indri, ISearch, MG4J, Namazu, Omega, OpenFTS, PLWeb, Solr, Sphinx, Terrier, Xapian, Zebra e Zettair, entre outros. Grande parte destes sistemas por um lado, não dão suporte à informação estruturada ou por outro, encontram-se descontinuados. Deste modo, irão ser abordados apenas os restantes sistemas: Indri, Omega, Solr, Sphinx e Zebra. Seguidamente, cada plataforma será descrita brevemente:

- *Apache Solr*[2]: plataforma de pesquisa por texto livre construída com o motor de pesquisa do projecto Apache Lucene. Tem uma comunidade muito forte, foi implementado na linguagem Java e tem uma base de utilizadores muito vasta;

[2] `http://lucene.apache.org/solr/`

- *Indri*[3]: sistema de pesquisa por texto construído a partir do projecto Lemur[4]. Este projecto foi desenvolvido através da cooperação entre a Universidade de Massachusetts e a Universidade Carnegie Mellon, nos Estados Unidos da América;

- *Omega*[5]: aplicação de pesquisa construída a partir do projecto Xapian[6]. Esta aplicação é desenvolvida na linguagem C++, mas tem conectores para várias linguagens (Perl, Python, PHP, Java, TCL, C#, etc);

- *Sphinx Search*[7]: motor de pesquisa desenhado para indexar informação estruturada e garantir indexação e pesquisa muito rápida. Este sistema é desenvolvido pela empresa Sphinx Technologies Inc.;

- *Zebra*[8]: motor de pesquisa que se apresenta como rápido, amigável e com muitas funcionalidades. Oferece APIs para várias linguagens de programação e permite indexar informação armazenada em vários formatos: XML/SGML, MARC, arquivos de e-mail, entre outros. Este software é mantido pela empresa Index Data[9].

Os sistemas tem variadíssimas características que os diferenciam. Cada sistema foi desenhado com vista a cobrir determinados cenários, logo para avaliar os sistemas há uma natural influência das necessidades de cada utilização. Assim para avaliar as várias plataformas deve ser definido um conjunto de factores quantificáveis e ortogonais a todas as elas: tempos de pesquisa e indexação, espaço ocupado pelo índice, funcionalidades, actividade da comunidade, suporte que oferece e ainda a licença que possui. A partir de vários *benchmarks* consultados [MBY07, Sin09, Hay04], é possível compilar a informação num quadro-resumo com os aspectos técnicos das plataformas, que pode ser consultado na Tabelas 2.8. Middleton et al. [MBY07] compara os sistemas Indri, Omega e Lucene (motor de pesquisa do Apache Solr), Singh [Sin09] testa os sistemas Lucene e Sphinx Search e Haye [Hay04] avalia Lucene, Xapian (motor de pesquisa do Omega) e Zebra. Como há pelo menos uma plataforma comum a todos os *benchmarks*, Lucene (Apache Solr), foi possível obter um termo de comparação entre todas as plataformas. De realçar, que foram tiradas conclusões semelhantes das plataformas que foram testadas em mais do que um *benchmark*.

Por outro lado, foi reunida informação referente a aspectos não técnicos das mesmas plataformas, igualmente importantes. Esta informação foi obtida consultando as comunidades, a documentação e as páginas de cada plataforma e está descrita na Tabela 2.8.

[3] http://www.lemurproject.org/indri/
[4] http://www.lemurproject.org/
[5] http://trac.xapian.org/wiki/Omega
[6] http://xapian.org
[7] http://sphinxsearch.com/
[8] http://www.indexdata.com/zebra
[9] http://www.indexdata.com/

Tabela 2.8: Quadro comparativo das especificações técnicas das tecnologias.

| **Plataforma** | **Indexação** | **Pesquisa** | **Tamanho do índice** | **Funcionalidades** |
|---|---|---|---|---|
| Apache Solr | Rápida | Rápida | Pequeno | Completo |
| Indri | Muito Rápida | Rápida | Normal | Algumas |
| Omega | Lenta | Lenta | Grande | Completo |
| Sphinx Search | Muito rápida | Muito rápida | Normal | Algumas |
| Zebra | Muito rápida | Rápida | Pequeno | Completo |

Tabela 2.9: Quadro comparativo do suporte que as tecnologias oferecem.

| **Plataforma** | **Documentação** | **Comunidade** | **Maturidade no mercado** | **Licença** |
|---|---|---|---|---|
| Apache Solr | Completa | Muito activa | Elevada (CNET, SourceFourge, ...) | Apache2 |
| Indri | Boa | Pouco activa | Baixa | BSD |
| Omega | Boa | Pouco activa | Baixa | GPL |
| Sphinx Search | Completa | Activa | Elevada (CraigsLists, Metacafe, ...) | GPL v2 |
| Zebra | Boa | Pouco activa | Baixa | GPL |

Uma vez que os tempos de resposta sobretudo na indexação e na pesquisa são essenciais, naturalmente que a plataforma Omega se destaca pela fraca classificação obtida [MBY07].

A plataforma Zebra alcança resultados muito bons [Hay04], no entanto, peca pelo suporte que tem, uma vez que tem uma comunidade pouco activa. As últimas actualizações têm ocorrido com pouca frequência e ainda se realça a fraca maturidade no mercado. Pela sua documentação, destaca-se ainda que a sua utilização é feita maioritariamente a nível académico.

O sistema Indri é bastante eficiente, com tempos de inserção e pesquisa ao nível dos melhores tempos de todas as plataformas [MBY07]. Para uma utilização comercial com alguma exigência, este software não tem uma comunidade nem maturidade no mercado que dê fortes garantias, uma vez que as contribuições ao projecto são feitas apenas por alunos das Universidades de Massachusetts e de Carnegie Mellon, nos EUA. Por exemplo, em projectos como o Apache Solr à uma grande comunidade activa à volta do sistema espalhada por todos os continentes. Deste modo, produzem-se com frequência novas actualizações e ainda são desenvolvidos vários projectos para estender e melhorar o Solr, tais como Apache Lucene, Apache Luke[10] e Apache Nutch[11].

Apache Solr e Sphinx Search são claramente os sistemas open-source com mais maturidade e aos quais são reconhecidas mais capacidades [Sin09] e serão detalhados seguidamente.

O projecto Apache Solr nasceu da carência de uma plataforma de pesquisa completa, visto que as soluções existentes à data da sua criação serem comerciais e com licenças

[10] `http://www.getopt.org/luke/`
[11] `http://nutch.apache.org/`

bastante caras [See06]. Este projecto é maduro e tem provas dadas em sítios *web* com a dimensão do AOL[12], NetFlix[13] e SourceForge[14], entre outros e tem como pontos fortes a rapidez de pesquisa e o poder de configuração. O Solr utiliza e estende a biblioteca de pesquisa Lucene [Fou09] para criar índices de texto e mecanismos de pesquisa. O código fonte do Solr é escrito em Java e corre como um servidor independente de pesquisa por palavras-chave. A comunicação com o servidor é feita de uma forma semelhante aos serviços *web*: enviando e recebendo documentos XML através do protocolo HTTP. Para utilizar o Solr é necessário criar uma aplicação que replique as transacções efectuadas numa base de dados para o servidor onde corre o Solr. As transacções inserir, actualizar ou apagar são comunicadas por pedidos HTTP POST através de documentos XML e as consultas à base de dados são comunicadas por pedidos HTTP GET [See06].

O Apache Solr oferece ainda um conjunto de *plugins*, um conjunto de programas que podem ser integrados, por exemplo o, Apache Nutch e uma poderosa configuração externa via XML de modo a que o sistema se adapte a qualquer tipo de aplicação sem que seja necessário estender o seu código Java [Fou06]. Um importante factor que abona a favor deste projecto é a comunidade activa que o desenvolve.

Sphinx Search é também um servidor de pesquisa por texto open-source. O código fonte é escrito em C++ e dá suporte aos sistemas operativos Linux, Windows, MacOS, Solaris, FreeBSD. Os seus pontos fortes são a eficiência, relevância nos resultados das pesquisas e simplicidade de integração [Ser09]. Este projecto é suportado pela empresa Sphinx Technologies Inc.

Tal como o sistema Apache Solr, o Sphinx Search foi concebido para ser escalável e constituir uma solução completa no que diz respeito aos motores de pesquisa por texto. A capacidade de personalização do Sphinx não é tão poderosa como a do Solr. Todavia, a Sphinx Technologies Inc oferece um conjunto vasto de serviços: suporte, consultoria, desenvolvimento e treino. Nem todos estes serviços são gratuitos, no entanto, garantem todo o apoio que possa ser necessário para adaptar este software open-source para qualquer aplicação [Ser09]. Comparativamente ao Solr, o Sphinx, de um modo geral, garante mais rapidez a nível da indexação e mais relevância na pesquisa, e ainda, oferece funcionalidades para a pesquisa estruturada [Aks10].

Para utilizar este software são disponibilizadas API oficiais para as linguagens PHP, Python, Java, Ruby e C [Str07] e API criadas pela comunidade do Sphinx para as linguagens C++, C#, Perl e Haskell.

[12] `http://www.aol.com/`
[13] `http://www.netflix.com/`
[14] `http://sourceforge.net/`

# Capítulo 3

# Descrição do Problema

O objectivo deste trabalho é sumariamente a definição de um sistema que, partindo de informação estruturada, permita a pesquisa de grandes volumes de informação de forma não estruturada. Na perspectiva do utilizador, a interacção que obtém quando utiliza um sistema de pesquisa sobre informação estruturada normalmente resulta num formulário com vários campos de texto que, na óptica do programador, gera consultas automaticamente para serem executadas num sistema de gestão de base de dados. Tal como sugere o título desta dissertação, um dos objectivos rígidos deste projecto é a conversão desses formulários, que têm várias caixas de texto, para um formulário com um único campo, ao estilo do que conhecemos da pesquisa nos motores de busca mais conhecidos. Tendo por base a melhoria de usabilidade nos formulários de pesquisa, os requisitos deste projecto também passam pela independência do esquema de base de dados e a eficiência na consulta de grandes volumes de dados.

## 3.1 Casos de Uso

Um sistema de base de dados permite a execução de inúmeras operações diferentes, que vão desde as operações para criação do esquema de base de dados (criação de tabelas, índices e chaves) às operações sobre a informação armazenada nas relações (apagar, inserir, alterar e pesquisar), entre outras. Neste projecto é impraticável, se tivermos em conta a sua duração, definir um sistema testando todas as operações possíveis que este oferece. Deste modo, é necessário focar num conjunto de operações mais recorrentes e exigentes e garantir que é feita uma análise intensiva sobre este conjunto. Primeiramente podemos analisar o CRUD, conjunto das operações básicas utilizadas em bases de dados relacionais: “actualizar”, “apagar”, “consultar” e “inserir”. No âmbito dos sistemas da PT Inovação duas das operações do CRUD têm uma frequência muito elevada: “inserir”

e “pesquisar” e as outras duas: “actualizar” e “apagar” ocorrem com uma frequência bastante menor. Se tomarmos em consideração um sistema que faz a gestão da informação de uma empresa de telecomunicações, as operações de tributação das chamadas, recarga dos telemóveis e consulta de saldo têm relevância máxima e um grau muito reduzido de tolerância. Estes três processos são constituídos basicamente por operações de inserção e consulta. Deve-se ainda realçar que estas duas últimas operações, por norma, são executadas na maior parte das vezes em tempo real, o que eleva ainda mais o grau de exigência perante outras executadas maioritariamente em situações de manutenção, como as operações de “actualizar” e “apagar”.

A PT Inovação faz ainda uso de algumas transacções mais específicas, mas de carácter relevante. Neste ramo destaca-se a operação de adição de um campo de texto a uma relação e consequente índice de texto sobre o novo campo. Este caso de uso corresponde à maior parte das alterações do esquema de base de dados de sistemas que já estejam em produção. Esta transacção adquire ainda mais relevância para a empresa uma vez que actualmente ao executar esta operação com a tecnologia de que dispõe, a eficiência da operação é muito baixa. Na Tabela 3.1 estão sumariados os casos de uso levantados.

Tabela 3.1: Casos de Uso.

| **Caso de Uso** | **Operações** |
|---|---|
| Inserir | Inserir informação na base de dados |
| Pesquisar | Efectuar consultas à base de dados |
| Indexar novo campo | Adicionar novo campo<br>Criar um índice no novo campo<br>Inserir em cada registo informação no novo campo |

Para simular um ambiente realista das bases de dados operadas pela PT Inovação é conveniente aproximar algumas grandezas aos ambientes de produção, nomeadamente o tamanho da base de dados e a frequência das operações já referidas. Mais precisamente, para obter uma boa aproximação, o número de registos das tabelas é frequente variar na ordem dos milhões e a frequência das transacções referidas pode ser aproximada por várias dezenas por segundo. Obviamente que esta informação é variável, mas para este projecto é importante definir um sistema que dê resposta aos sistemas em produção mais exigentes da PT Inovação.

## 3.2 Tecnologias

A empresa PT Inovação tem bastante experiência em base de dados tradicionais, pelo que seria útil avaliar as potencialidades dos sistemas desta natureza, uma vez que apenas lhes é reconhecida, universalmente, a capacidade para responder a pesquisas estruturadas.

Tomando em consideração a revisão das tecnologias, que pode ser consultada no Capítulo 2, não se pode ignorar a oferta existente relativamente ao software open-source criado para tornar a informação estruturada acessível a partir de pesquisas por palavras-chave. Tal como referido no mesmo capítulo, a semântica destas tecnologias varia bastante entre orientação ao documento ou ao registo e foco na indexação ou no armazenamento. Foi decidido explorar um sistema representativo do âmbito relacional e uma plataforma que se baseie num modelo não relacional.

### 3.2.1 Sistema relacional

Os sistemas de gestão de base de dados relacionais mais conhecidos suportam a pesquisa por palavras-chave. Podemos destacar Oracle através da extensão Oracle Text [Cor09b], IBM DB2 com o módulo DB2 Text Search [Cor10] e ainda Microsoft SQL Server [Lop08], MySQL [Kar09] e PostgreSQL [BS09] que oferecem este serviço nativamente. Para aproveitar da melhor forma a experiência nestas tecnologias da empresa PT Inovação, foi decido adoptar o SGBD que a empresa utiliza, o Oracle. Assim, este sistema que é líder no mercado será seleccionado como uma solução candidata relacional à pesquisa livre por texto em informação estruturada.

### 3.2.2 Sistema não relacional

A pesquisa por palavras-chave foi dada a conhecer em grande parte pelos motores de busca mais conhecidos. Rapidamente conquistou popularidade e, nos últimos anos foi dedicado um esforço considerável quer da parte de particulares, como o caso da empresa Sphinx Technologies Inc, quer da parte de organizações de software livre, como a Apache, quer a nível académico, como aconteceu com o projecto Omega.

A partir da revisão das tecnologias efectuada no Capítulo 2 constata-se que há claramente dois sistemas que se destacam: Apache Solr e Sphinx Search. Entre os dois sistemas, temos vantagens de um lado e do outro e para seleccionar um dos sistemas é necessário estabelecer prioridades. Ambas as plataformas têm um estatuto reconhecido no mercado, oferecem garantias de suporte pelas suas comunidades activas e ainda suporte comercial pago. Por um lado, o Sphinx leva clara vantagem perante a concorrência a nível de eficiência e dá a garantia de ser o software mais rápido neste ramo [Aks10]. Por outro lado, Solr oferece muitas funcionalidades importantes que Sphinx não suporta, como por exemplo: pesquisas facetadas (*faceted search*), corrector ortográfico e remoção de duplicados nos resultados (*field collapsing*) [See06]. Entrando em detalhes mais técnicos, temos uma integração directa do Sphinx com MySQL e PostgreSQL, mas tanto Sphinx como Solr permitem uma integração com os SGBD via conectores ODBC [Ser09]. O sistema Apache Solr oferece uma API para a linguagem Java, um conjunto de *plugins* e uma

poderosa configuração externa via XML. Já para utilizar o Sphinx, são apenas disponibilizadas API para várias linguagens de programação, o que acaba por ser uma limitação em relação a Solr, visto que podendo ser configurado por XML, a sua interacção está virtualmente ao alcance de qualquer linguagem. Uma última comparação pode ser feita através das licenças que têm. A licença Apache2 da plataforma Solr permite utilizar o software sem adquirir uma licença, ao contrário da licença GPL v2 do Sphinx.

Pesando os factores enunciados no parágrafos anteriores, a decisão tomada neste trabalho foi o uso da plataforma Apache Solr. A justificação prende-se com vários factores: a licença Apache2, o suporte da activa comunidade da Apache e a vasta gama de funcionalidades oferecidas.

# Capítulo 4

# Metolodogia

Este capítulo descreve o método e o trabalho realizado com vista à definição de um sistema que permita a pesquisa de texto livre sobre informação estruturada, a partir das tecnologias seleccionadas no Capítulo 3.

## 4.1 Abordagem

Para concretizar o objectivo desta dissertação, foi decidido criar uma bateria de testes baseada nos casos de uso discutidos no capítulo anterior que colocasse à prova um conjunto de tecnologias escolhidas de entre aquelas que foram estudadas no Capítulo 2. Como já foi referido no Capítulo 3, será posto à prova o SGBD relacional Oracle e a tecnologia open-source Solr. Oracle é mais vocacionado para o armazenamento da informação e Solr está mais preparado para a indexação. Assim, e de modo a obter resultados justos e conclusivos é essencial balancear os testes de modo a não beneficiar ou prejudicar alguma plataforma.

Para comparar as plataformas escolhidas, foi criado um conjunto de testes com o intuito de analisar o desempenho das tecnologias quer a nível temporal, quer a nível de espaço quando postas à prova num determinado conjunto de cenários que simulem ambientes realistas de produção. Os testes foram ainda concebidos para prever, de um modo fiável, como se comportam as tecnologias quando se faz variar determinado parâmetro em cada caso de uso.

## 4.2 Ambiente de Testes

Nesta secção será enunciado o ambiente onde correram os testes quer a nível de hardware, quer a nível de software. Infelizmente alguns recursos escasseiam, e de facto, não

foram criadas condições para simular o ambiente perfeito para efectuar os testes. Mais precisamente, cada plataforma idealmente executaria numa máquina física [Not08], mas apenas foram disponibilizadas máquinas virtuais para correr cada uma das plataformas. A arquitectura de uma máquina virtual, implica a adição de mais uma camada de abstracção e não permite ter controlo sobre a globalidade do sistema [SN05]. Estas condicionantes diminuem a fiabilidade dos testes.

Cada plataforma e aplicação cliente foram instaladas em máquinas virtuais diferentes, que se encontram no mesmo computador. A nível de *hardware*, o computador em que as máquinas virtuais se encontram tem um processador Dual Core Intel Xeon X5670 2.93GHz e 2 GBytes de memória RAM. Existem ainda 2 GBytes de memória secundária.

Quanto a *software*, o sistema operativo instalado é o Red Hat Enterprise Linux Server versão 5.4 e a versão de Java utilizada é a 1.6.0_24. Ambas as plataformas usufruem destas configurações. As versões de Oracle e Solr instaladas são respectivamente as versões 11.2.0.2 e 1.4.1.

## 4.3 Considerações sobre os Testes

Para eficazmente comparar as plataformas, pretende-se neste projecto criar um ambiente de execução o mais equilibrado possível de forma a obter resultados justos. Por um lado, em Solr não existe o conceito de normalização, originado nas base de dados relacionais, o que constitui uma condicionante para o esquema de base de dados a utilizar. O esquema desenhado para os testes consiste numa única relação, com um vasto conjunto de atributos e pode ser pensado como uma base de dados desnormalizada, como exemplificado na Tabela 2.7. Por outro lado, em Solr existe a possibilidade de armazenar ou não cada atributo. Em Oracle, não existe este conceito. Cada valor inserido num atributo é armazenado e pode ser sempre retribuído num resultado de uma consulta. Para efeitos deste trabalho, esta funcionalidade do Solr é ignorada e todos os valores inseridos são armazenados. Entende-se assim que com a mesma configuração, os sistemas serão postos à prova nas mesmas condições. Obviamente que seria interessante explorar as funcionalidades de ambas as plataformas, mas o método seguido neste trabalho consiste em colocar os sistemas num estado em que possam ser comparados, o que envolve a utilização de uma configuração básica em ambos.

Para avaliar os sistemas recorrendo a um conjunto de testes, é necessário contemplar a presença de situações indesejáveis, mas impossíveis de controlar, como por exemplo a variação da carga do processador ao longo de cada execução. Para tal, e para eliminar *outliers*, o resultado de cada teste criado é calculado através da média de pelo menos 30 execuções do referido teste em cada plataforma. Assim e recorrendo ao Teorema do Limite Central, garantimos que a distribuição tende a ser normal, ao calcular a média de um número significativo de amostras, independentemente do tipo de distribuição das

medições efectuadas [LNT00a]. Por outro lado, sempre que possível os testes foram executados em diferentes alturas do dia.

Outro factor que pode condicionar os testes é a linguagem de programação com que foram implementadas as plataformas, neste caso a linguagem Java. O processo de compilação de programas Java envolve a compilação do código fonte para uma representação intermédia, apelidada de *bytecodes*, o que permite ao Java ser uma linguagem multiplataforma. Quando uma aplicação Java é executada numa máquina os *bytecodes* são executados [SOT+00]. A partir desta representação, e uma vez que interpretar os *bytecodes* é um processo lento, por não serem optimizados para nenhuma arquitectura em específico, é utilizado o compilador Java JIT (*just-in-time*) que faz a conversão dos *bytecodes* em código nativo da máquina em que executa [SOT+00]. Um dos principais problemas deste compilador é que no início da execução é necessário carregar em memória e compilar os *bytecodes*, o que torna a fase inicial de execução um pouco mais lenta [SOT+00].

Especificamente neste trabalho, ao correr um teste é normal que as primeiras execuções, que equivalem às menos exigentes, possam ser substancialmente mais lentas que as seguintes. Para evitar este problema, no início de cada teste dá-se lugar a uma fase de "aquecimento", onde são feitas várias execuções sem serem registadas, para que todas as execuções de cada teste sejam feitas nas mesmas condições. No entanto, apesar de tudo é natural que mesmo assim se sinta o referido efeito nos resultados obtidos.

## 4.4 Bateria de Testes

A bateria de testes foi desenhada conforme os casos de uso enunciados no Capítulo 3. A ideia principal é a simulação de um ambiente em produção dos sistemas operados pela PT Inovação.

Como já referido na subsecção anterior, devido à inerente orientação ao "documento" da plataforma Solr, o esquema da base de dados a utilizar não estará normalizado e consistirá numa tabela (ou documento) com um determinado conjunto de campos de texto e campos inteiros. Visto que a relevância deste trabalho é a indexação e pesquisa de campos de texto, não há necessidade de contemplar na base de dados outros tipos de atributos.

Para permitir uma pesquisa rápida por texto é necessário indexar os campos de texto pelos quais pretendemos pesquisar. Por um lado, esta decisão varia de problema para problema e ao mesmo tempo o requisito da independência do esquema da base de dados não deve ser ignorado. Por outro lado, Solr é claramente uma plataforma mais vocacionada para a indexação de campos de texto comparativamente a Oracle. Oracle por seu lado tem uma maior propensão para o armazenamento, interrogação e processamento. Por estas razões, foi decidido equilibrar os testes criando sempre três cenários para cada situação: indexação de um campo de texto, indexação de metade dos campos de texto e a indexação de todos os campos de texto.

Neste trabalho, o conceito de teste corresponde à execução de um determinado número de vezes da aplicação cliente. Por cada execução, a aplicação efectua uma tarefa dependente do caso de uso a que se refere o teste. Entre execuções do mesmo caso de uso, um determinado factor varia. Por fim, a aplicação regista as latências de execução da tarefa, entre outros valores dependentes do teste. Cada teste ainda é repetido um mínimo de 30 vezes para serem analisados comportamentos mais fiáveis, como referido na Secção 4.3.

Os testes criados para os três casos de uso foram estruturados de forma diferente. Em seguida, serão detalhadas as configurações usadas para cada teste.

### 4.4.1 Caso de uso "inserir"

Relativamente ao primeiro caso de uso, a informação vai ser inserida em *batches* para as bases de dados, cada um contendo um conjunto fixo de registos. Identificam-se vários factores que potencialmente podem condicionar o desempenho das plataformas ao inserir a informação na base de dados. Assim sendo, foi desenhado um conjunto de testes que pretende identificar como se comporta o sistema em determinada situação:

- Teste dos Batches – acumulação de informação na base de dados;
- Teste dos Registos – recepção e indexação de um grande conjunto de dados, ou seja, um *batch* com um grande número de registos;
- Teste dos Campos de Texto – variação do número de campos de texto da base de dados;
- Teste dos Campos Indexados – variação do número de campos de texto indexados;
- Teste dos Campos de Tipo Inteiro – variação do número dos campos do tipo inteiro;
- Teste dos Índices de Texto – acumulação de informação na base de dados, utilizando índices de texto.

Em cada teste será apenas posto à prova um único factor (número de registos na base de dados, número de atributos de texto, etc). Os primeiros cinco testes referidos nesta secção foram preparados para os índices por omissão das plataformas, ou seja, para índices que indexam o campo por inteiro e não permitem que a pesquisa parcial seja eficiente. O Teste dos Índices de Texto foi concebido para analisar a eficiência das plataformas na pesquisa de palavras em texto e foi desenhado apenas para o cenário intermédio, onde são indexados metade dos campos de texto.

É necessário ainda definir uma configuração padrão ortogonal a todos os testes que vão ser efectuados. A configuração que foi estabelecida para todos os testes do caso de uso "inserir" pode ser consultada na Tabela 4.1.

Tabela 4.1: Configuração de cada teste pertencente ao caso de uso "inserir".

| **Teste** | ***Batches*** | **Registos** | **Cam. de texto** | **Cam. indexados** | **Cam. de inteiros** |
|---|---|---|---|---|---|
| T. dos Batches | Varia | 250 | 20 | 1, 10, 20 | 5 |
| T. dos Registos | 1 | Varia | 20 | 1, 10, 20 | 5 |
| T. dos Cam. de Texto | 1 | 250 | Varia | 1, metade, todos | 5 |
| T. dos Cam. Indexados | 1 | 250 | 1000 | Varia | 5 |
| T. dos Cam. Inteiros | 1 | 250 | 20 | 1, 10, 20 | Varia |
| T. dos Índices de Texto | 1 | Varia | 20 | 1, 10, 20 | 5 |

Relativamente ao Teste dos Campos Indexados, não se aplicam os três cenários de indexação, visto que se pretende aumentar gradualmente o número de campos a indexar para graficamente prever o comportamento do sistema. Merece ainda destaque o facto dos Testes dos Campos de Texto, dos Campos Indexados e dos Campos de Tipo Inteiro estarem limitados pela plataforma Oracle, que permite criar para cada relação um máximo de um milhar de atributos.

### 4.4.2 Caso de uso "pesquisar"

O caso de uso "pesquisar" é composto por três partes: criação da estrutura da base de dados, inserção de informação e execução de um conjunto de consultas à base de dados. O processo de executar as consultas à base de dados consiste na criação de um conjunto de *threads* para simular a concorrência nos pedidos, em que cada *thread* tem a seu cargo um conjunto de pesquisas a efectuar durante um segundo. Os resultados apenas são contabilizados no caso de cada *thread* completar em cada segundo o número de operações que lhe compete. São utilizadas 10 *threads* em todos os testes. Devido às implementações das duas plataformas não é possível distinguir os tempos de envio da consulta, de espera pela resposta e recepção dos resultados. Assim, cada execução regista a latência total desde o início do envio da pesquisa até ao fim da recepção dos resultados.

Neste caso de uso temos factores que condicionam o desempenho das plataformas ao pesquisar a informação na base de dados com origem na informação que foi inserida na base de dados, no tipo e na forma como são feitas as consultas. Deste modo, os testes criados e os cenários que examinam são os seguintes:

- Teste dos Registos – pesquisa de frases com a presença de muitos registos na base de dados;
- Teste dos Campos de Texto – variação do número de campos de texto da base de dados (indexados ou não);
- Teste dos Campos a Pesquisar – variação do número de campos de texto a indexar e a pesquisar;

- Teste dos Resultados – variação do número de resultados a retornar em cada consulta;
- Teste da Carga – variação da carga na plataforma por via do aumento do número de consultas a realizar por segundo;
- Teste dos Índices de Texto – pesquisa de palavras com a presença de muitos registos na base de dados.

Neste trabalho, para um campo ser pesquisável tem de estar indexado, uma vez que a dimensão das bases de dados com que foram feitos os testes obriga à criação de um índice para se pesquisar eficientemente.

Os primeiros cinco testes referidos nesta secção foram preparados para os índices por omissão das plataformas, que indexam o campo por inteiro e não permitem a pesquisa parcial. No entanto, à imagem do que sucedeu na secção anterior pretende-se testar a pesquisa de palavras em texto. Em Oracle, é possível utilizar vários tipos de índices: B-Tree (índice utilizado por omissão), Bitmap, CONTEXT e CTXCAT, entre outros. Os dois primeiros não permitem eficientemente a pesquisa de uma palavra em texto, como foi discutido na Secção 2.2, e têm como diferença o facto do índice Bitmap ser optimizado para trabalhar com atributos de baixa cardinalidade. Os dois últimos índices pertencem ao módulo Oracle TEXT e permitem a pesquisa de palavras em texto. O índice CONTEXT é mais vocacionado para grandes quantidades de texto e o índice CTXCAT é optimizado para pesquisar em textos curtos [Cor09b]. Já em Solr não é possível escolher o índice aplicado, apenas há a possibilidade de incorporar filtros no índice [Cut05]. Entre os filtros disponíveis destaca-se o filtro divisor por espaços (*whitespace tokenizer*), que permite a pesquisa parcial de um atributo. É importante para este trabalho aferir as capacidades de cada índice e à imagem do que foi enunciado no caso de uso "inserir", foi desenhado o Teste dos Índices de Texto para se averiguar a tendência das pesquisas com os vários índices à medida que se varia a quantidade de informação armazenada. Este teste apenas será executado para o cenário intermédio, que corresponde a indexar e pesquisar metade dos campos de texto. Um aspecto importante na pesquisa por palavras é a fonte de palavras a pesquisar. Será interessante verificar se há diferença de comportamento ao pesquisar conjuntos de palavras com frequências distintas na base de dados. Assim, este teste será repetido para cada os vários conjuntos de palavras definidos (Secção 4.5): as palavras raras, frequentes e muito frequentes. Esta informação pode ser consulta na Tabela 4.2.

A configuração que foi estabelecida para todos os testes do caso de uso "pesquisar" encontra-se listada na Tabela 4.3. Uma nota para os Testes dos Campos a Pesquisar e dos Índices de Texto onde apenas se executam 10 operações por segundo. Estes testes são os mais exigentes e para que não fiquem limitados pelo número de operações por segundo, foi decidido baixar este factor para que as suas variáveis tenham uma maior amplitude.

Tabela 4.2: Configuração do Teste dos Índices de Texto.

| Plataforma | Tipo de índice | Dicionários | Campos a pesquisar |
|---|---|---|---|
| Oracle | CONTEXT | Raras | 10 |
| | CTXCAT | Frequentes | |
| Solr | Normal (*whitespace tokenizer*) | Muito Frequentes | |

Tabela 4.3: Configuração de cada teste pertencente ao caso de uso "pesquisar".

| Teste | Registos | Cam. de texto | Cam. a pesquisar | Resultados | Ops por seg |
|---|---|---|---|---|---|
| T. dos Registos | Varia | 20 | 1 , 10 , 20 | 20 | 50 |
| T. dos Cam. de Texto | 200.000 | Varia | 1 , metade , todos | 20 | 50 |
| T. dos Cam. a Pesquisar | 100.000 | 40 | Varia | 20 | 10 |
| T. dos Resultados | 200.000 | 20 | 1 , 10 , 20 | Varia | 50 |
| T. da Carga | 200.000 | 20 | 1 , 10 , 20 | 20 | Varia |
| T. dos Índices de Texto | Varia | 20 | 10 | 20 | 10 |

### 4.4.3 Caso de uso "indexar novo campo"

O caso de uso "indexar novo campo" é desenrolado de forma diferente nas duas plataformas. Em Solr, a estrutura dos documentos é dinâmica, ou seja, não é necessário alterar a sua estrutura para adicionar um novo atributo a cada documento. No entanto, em Oracle o processo é mais complexo: primeiro é necessário adicionar um campo à estrutura da tabela e depois é preciso criar um índice no novo campo. Por fim, e comum a ambas as plataformas, é necessário fazer uma actualização a cada registo (ou documento) preenchendo o novo atributo. A aplicação cliente mede em cada execução a latência total deste processo. Ao considerarmos a diferença na implementação deste caso de uso entre os dois sistemas é importante ter a percepção em Oracle das latências individuais de cada tarefa: alterar estrutura da tabela, criar índice e adicionar informação.

Este teste será feito por forma a observar como se comporta o sistema com o aumento de registos na base de dados e será feito apenas para o cenário intermédio. A configuração deste teste é apresentada na tabela 4.4.

Tabela 4.4: Configuração do teste referente ao caso de uso "indexar novo campo".

| Atributo | Valor |
|---|---|
| Número de registos | Varia |
| Campos de texto | 20 |
| Campos indexados | 10 |
| Campos do tipo inteiro | 5 |

## 4.5 Informação a inserir na base de dados

Uma parte que não pode ser menosprezada neste processo de *benchmarking* é a informação a ser inserida nas estruturas de dados. Assim, e à imagem do que foi feito com o esquema de base de dados, a informação a inserir deve ser aleatória para não haver qualquer dependência nos resultados obtidos. Para além da aleatoriedade, os dados foram gerados de forma diversificada para não viciarem os índices das estruturas e estarem próximos da realidade. Seguindo esta ideia, foram geradas 900 palavras de comprimentos a variar entre 3 e 11, inclusive (uma centena por cada comprimento). As palavras foram seleccionadas aleatoriamente e na mesma proporção para um de três conjuntos (três centenas para cada conjunto): muito frequentes, frequentes e raras. A partir destas palavras foram construídas frases, que equivalem a conjuntos de palavras separadas por espaços. As frases são constituídas aleatoriamente por conjuntos de 1 a 5 palavras. O processo de escolha das palavras para cada frase consiste em:

- 60% dos casos seleccionar uma palavra do conjunto das palavras muito frequentes;
- 30% dos casos seleccionar uma palavra do conjunto das palavras frequentes;
- 10% dos casos seleccionar uma palavra do conjunto das palavras raras.

Com este processo foram geradas cem mil frases que foram registadas num ficheiro. Em cada teste, as aplicações cliente devem iterar sobre este ficheiro à medida que necessitem de uma frase para inserir ou pesquisar.

Com este processo, ambas as aplicações inserem e pesquisam exactamente a mesma informação, que foi gerada de forma aleatória.

## 4.6 Plataforma de Testes

A implementação prática deste trabalho consistiu na criação de duas aplicações cliente na linguagem Java para testarem as instalações de Oracle e Solr. Cada aplicação cliente recebe um conjunto de parâmetros de entrada obrigatórios que referenciam o teste a efectuar e o cenário escolhido. As aplicações estão ainda preparadas para receber outros parâmetros opcionais, dependentes de cada teste. Tendo em conta o grande conjunto de variáveis envolvidas nos testes, para evitar o trabalho de passar as variáveis nos parâmetros, os seus valores encontram-se definidos num ficheiro de configuração. As aplicações foram criadas de forma a minimizar as diferenças na implementação, mas como as API das plataformas são diferentes, naturalmente em algumas situações as implementações são necessariamente diferentes.

### 4.6.1 Solr

Nesta subsecção será descrita a implementação de Solr, começando pelo ambiente de execução. O sistema Solr necessita de um contentor Servlet para executar. Para efeitos deste trabalho, foi utilizado um Servlet criado pela mesma organização, o Apache Tomcat, versão 6.0.32 [Fun11]. A versão de Solr instalada é a 1.4.1.

Em Solr, a especificação do esquema de base de dados é feita num ficheiro de configuração XML no Servlet. A configuração utilizada na maioria dos testes inclui campos de texto indexados, campos de texto não indexados e campos inteiros. O número de campos pode variar, no entanto, com uma única definição se pode contemplar estas situações. Recorrendo à funcionalidade de *dynamic field*, podemos especificar que todos os campos com um determinado sufixo têm um determinado tipo. Deste modo os três tipos de campos criados são:

- **_text_indexed*: para campos de texto indexados;
- **_text_not_indexed*: para campos de texto não indexados;
- **_integer*: para campos inteiros indexados;

A aplicação cliente faz uso da API Java oficial SolrJ [Fou11], versão 1.3. Para comunicar com a instância de Solr cria-se um objecto *CommonsHttpSolrServer* que implementa as funções de *add* (adicionar documentos) e de *query* (consulta documentos). Em Solr, se for adicionado um documento com um identificador repetido, o documento original é sobrescrito. Assim, o método *add*, que recebe um *batch* de documentos, executa operações de inserção e actualização. O método *query* recebe um objecto *SolrQuery*, que por sua vez recebe uma string de pesquisa. Em Solr, caso não seja especificado um campo a pesquisar, a pesquisa é feita por um campo por omissão. No entanto, graças à funcionalidade de *copyField* podemos definir que a pesquisa é feita sobre um certo conjunto de campos. Neste trabalho, os campos a pesquisar são todos os campos de texto indexados.

Nos testes feitos à plataforma Solr para registar o tamanho da estrutura de dados resultante é utilizada a ferramenta Luke [Bia08]. Esta ferramenta open-source permite inspeccionar índices criados pelos sistemas Solr e Lucene.

### 4.6.2 Oracle

A versão de Oracle instalada utilizada foi a 11.2.0.2. A ligação da aplicação cliente com a instalação de Oracle foi feita recorrendo ao *driver* OJDBC (Oracle Java Database Connectivity), versão 6. Em cada teste é criada uma tabela com um conjunto de campos de texto e campos do tipo inteiro. Os campos de texto são definidos como *varchar2* e é necessário especificar o seu tamanho. O tamanho do campo foi definido a 60 caracteres, para garantir que não é inferior ao comprimento máximo dos valores de entrada, como

pode ser confirmado na Secção 4.5 deste capítulo. Depois vem a criação dos índices sobre um conjunto de campos de texto. Para inserir e pesquisar é preciso criar um objecto *CallableStatement*, que permite inserir e executar consultas em *batch*. Este objecto recebe como parâmetro a transacção em linguagem SQL a efectuar. Se uma inserção é semelhante a Solr, onde é necessário especificar para cada atributo que valor é inserido, já para consultar as tecnologias têm implementações bastante diferentes. Em Oracle, é necessário colocar a lista de campos por onde se pretende pesquisar e estes devem estar separados pela estrutura condicional *OR* para que um registo seja seleccionado no caso de pelo menos um dos campos verificar a condição.

Nos testes onde se regista o tamanho da tabela criada, é executada uma consulta que devolve a soma do comprimento de todos os segmentos alocados à tabela. Para tal, recorre-se à tabela DBA_SEGMENTS que descreve o espaço alocado para todos os segmentos na base de dados [Cor09a].

# Capítulo 5

# Resultados Experimentais

Este capítulo descreve os resultados experimentais obtidos com o trabalho efectuado seguindo a metodologia apresentada no Capítulo 4. O objectivo principal das experiências realizadas é a comparação das tecnologias e previsão dos seus comportamentos em ambientes realistas. No fim deste capítulo estão documentadas as conclusões gerais e cenários possíveis de utilização destas tecnologias.

Para cada caso de uso foi executado um conjunto de testes e por cada teste foi gerado pelo menos um gráfico. Nas três secções seguintes são apresentadas análises detalhadas sobre os gráficos gerados.

## 5.1 Caso de uso "inserir"

O caso de uso "inserir" está dividido em seis testes diferentes. Nos primeiros cinco são utilizados os índices por omissão de cada plataforma e no último teste é feita uma repetição do Teste dos Registos, mas utilizando índices que façam a divisão de texto em palavras.

### 5.1.1 Teste dos Batches

Neste teste o objectivo é confirmar a tendência de Solr e Oracle a acumular dados na base de dados. A Figura 5.1 apresenta-nos a relação do aumento do número de *batches* inseridos na base de dados com o tempo médio de inserção de cada *batch*. Cada *batch* é composto por 250 registos. Este gráfico diz respeito ao Cenário I, onde é indexado 1 campo de texto em 20.

Um ponto que pertença à linha vertical dos 2000 *batches*, destacada na Figura 5.1, indica que foram inseridos consecutivamente 2000 *batches* de 250 registos na base de dados. No caso do Solr, como o ponto de intercepção da curva com essa recta se situa

nos 88 milisegundos, significa que a média dos tempos de inserção de cada *batch* dos 2000 inseridos é de 88 milisegundos. Cada ponto no gráfico resulta, tal como em todos os gráficos, numa média de 30 execuções. Através da observação do gráfico conclui-se que as curvas estabilizam nos 89 milisegundos no caso do Solr e nos 23 milisegundos no caso do Oracle, aproximadamente a partir dos 1500 *batches* inseridos. Ou seja, Oracle demora aproximadamente um quarto do tempo de Solr a inserir e indexar cada *batch*, o que se pode considerar uma diferença significativa.

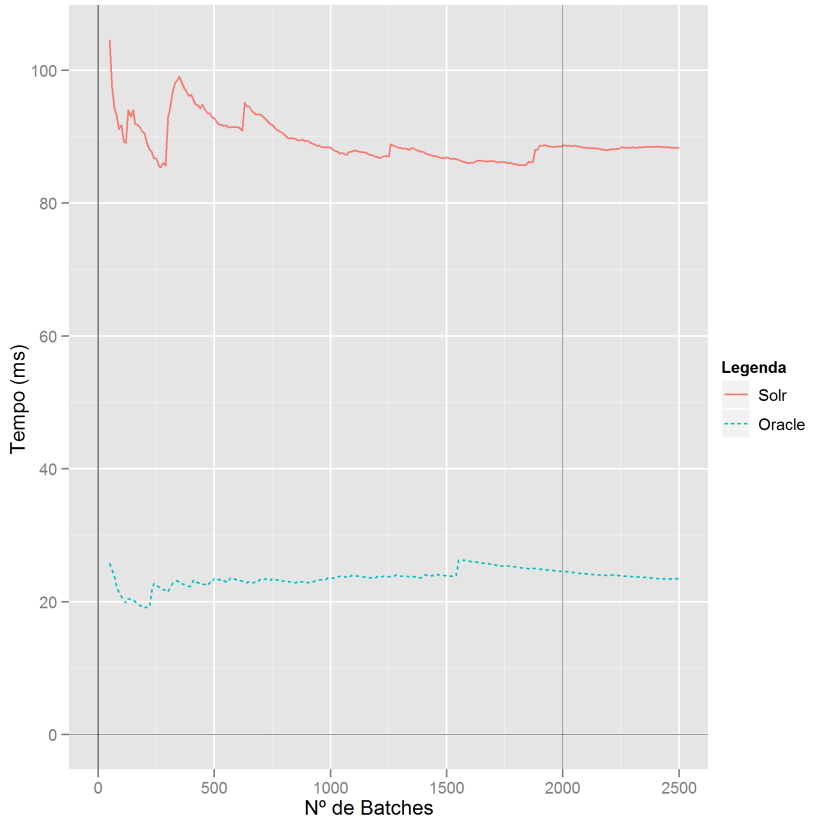



Figura 5.1: Latência média de inserção dos *batches* à medida que a informação acumula na base de dados, referente ao Cenário I.

Avançando para o cenário seguinte, onde são indexados metade dos 20 campos de texto, é beneficiada a plataforma Solr como se comprova pela Figura 5.2. Solr tem um desempenho muito bom na indexação de grandes conjuntos de atributos como podemos observar ao longo deste caso de uso. Naturalmente surge uma degradação considerável de performance em Oracle, mas quanto ao Solr, as latências aumentam 50% no global relativamente ao cenário anterior, o que é um aumento insignificante se considerarmos que estão a ser indexados mais 9 campos de texto. Neste caso, a curva referente ao Solr é mais lenta no início do teste, no entanto, apresenta uma elevação mínima quando comparada com Oracle, que a partir da marca dos 900 *batches* aumenta exponencialmente o tempo

médio de inserção de cada *batch*. Neste trabalho Oracle, tal como Solr, é utilizado com a sua configuração de base e não são utilizados processos ou alterações na configuração com vista à melhoria de performance. Todavia, a instalação de Oracle em ambientes exigentes exige que seja feito um acompanhamento das estatísticas que vai gerando [Cor03]. Mais detalhadamente, Oracle determina o plano de operações a efectuar em cada transacção com base em estatísticas da base de dados que devem ser calculadas periodicamente ou em situações excepcionais, como por exemplo quando é alterado o esquema da base de dados ou quando é inserido um grande conjunto de registos [Cor03]. Oracle serve-se da informação das estatísticas que calcula para diversos fins, tais como agendamento de reconstruções dos índices ou até para determinar se devem ou não ser indexados um conjunto de registos, consoante seja benéfico [Cor03, Bur11]. Considerando a curva exponencial de Oracle, que pode ser vista na Figura 5.2, conclui-se que Oracle necessita claramente de uma reconstrução ou optimização dos índices. Os motivos da perda de performance dos índices à medida que vão indexando mais informação prendem-se com uma natural fragmentação do índice e eventual perda de selectividade ao deformar a árvore que mantêm [Bur11].

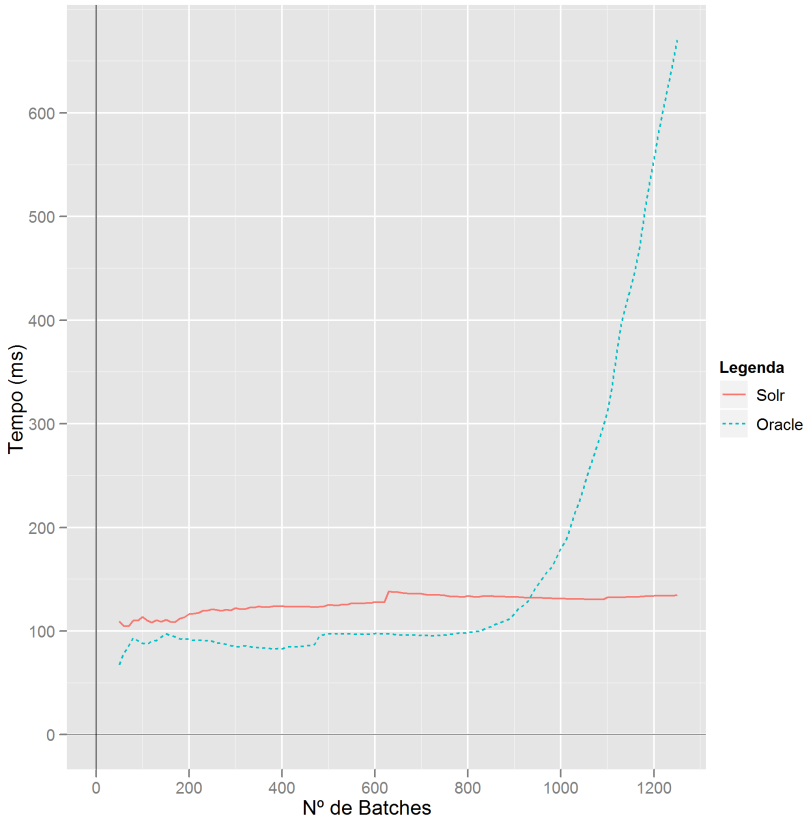



Figura 5.2: Latência média de inserção dos *batches* à medida que a informação acumula na base de dados, referente ao Cenário II.

Passando para o Cenário III, confirma-se a tendência observada na passagem do primeiro cenário para o segundo cenário, como pode ser observado na Figura 5.3. O mesmo é dizer que o desempenho de Oracle é muito mais afectado que o desempenho de Solr, com o aumento do número de atributos indexados. À imagem do cenário anterior, Solr inicia o teste com uma latência média próxima à de Oracle, no entanto, a partir dos 400 *batches* Oracle distancia-se largamente de Solr. Note-se que a curva registada se refere à latência média de cada *batch*. Assim e uma vez que o gráfico é exponencial, fica ainda mais destacado o fraco desempenho de Oracle na parte final do teste, aproximadamente a partir dos 550 *batches*.

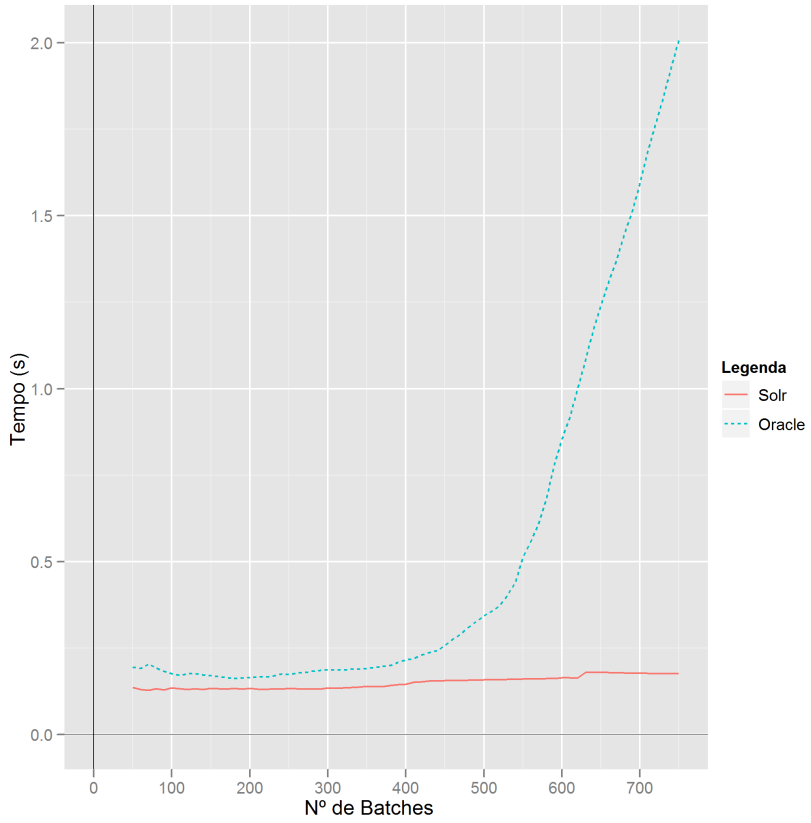



Figura 5.3: Latência média de inserção dos *batches* à medida que a informação acumula na base de dados, referente ao Cenário III.

Após a análise temporal, podemos aferir o comportamento das plataformas a nível espacial, através da Figura 5.4. Ao longo do presente capítulo serão apresentados diversos gráficos sobre a análise do espaço ocupado pelos índices criados das plataformas. A estrutura dos gráficos segue os mesmos moldes: as curvas apresentadas são a proporção da dimensão da estrutura de dados criada por Solr relativamente à estrutura criada por Oracle. Cada curva representa cada cenário de execução: indexação de um campo de texto, metade dos campos de texto ou todos os campos de texto. Sempre que a curva

estiver por baixo da recta dos 100% significa que a estrutura criada por Solr tem dimensão inferior à estrutura criada por Oracle e vice-versa.

Constata-se à primeira observação que as curvas nos três cenários são muito semelhantes, apenas estando deslocadas na vertical. Comparando as duas plataformas, temos que, no geral Solr cria uma estrutura com uma dimensão inferior à do seu homólogo e que se acentua à medida que o número de campos indexados aumenta. Apesar desta conclusão, a proporção do índice de Solr praticamente não baixa dos 80% da estrutura de Oracle. Outra curiosidade que se destaca no gráfico é o efeito de "degraus". Para explicar este efeito vamos recorrer aos gráficos que relacionam o tamanho do índice criado, com o aumento do número de *batches* inseridos individualmente para cada plataforma.

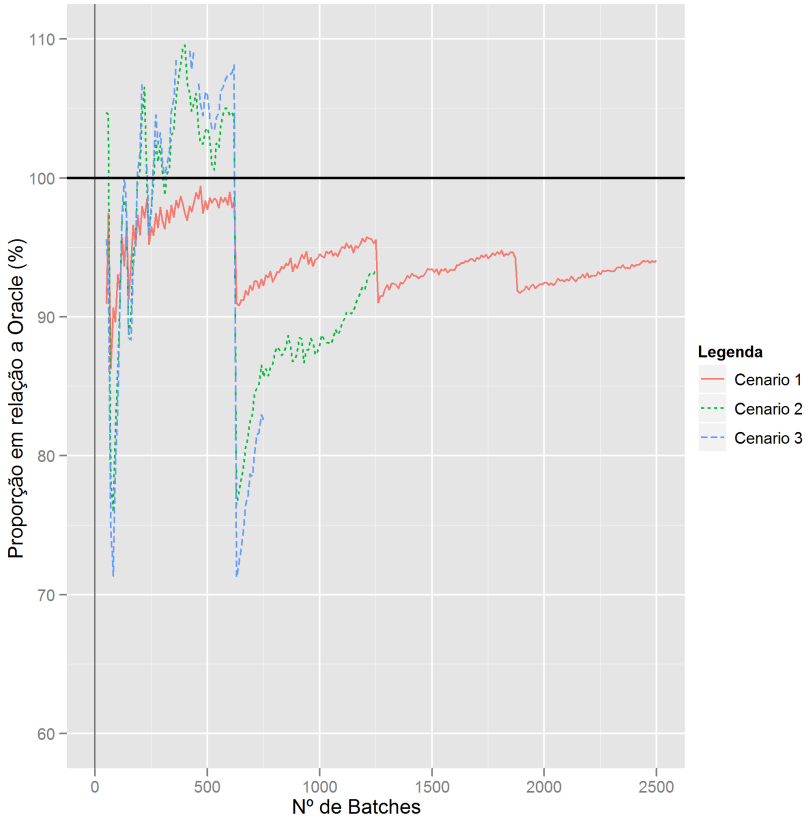
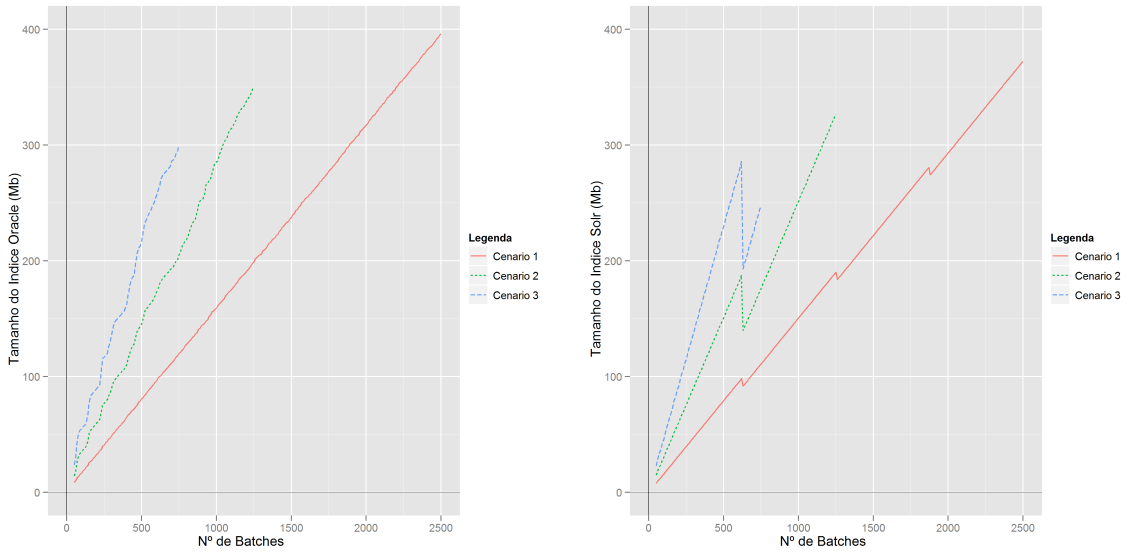



Figura 5.4: Comparação das estruturas de dados geradas em Solr, comparativamente a Oracle nos três cenários referentes ao Teste dos Batches.

Na Figura 5.5 temos lado-a-lado o crescimento dos índices em Oracle e Solr, à medida que acumulam informação na base de dados. Constata-se de um modo claro que a nível de espaço, a estrutura em Oracle tem um crescimento linear e bastante regular, à medida que acumula informação. A diferença entre os três cenários prende-se com a inclinação da rectas, ou seja, no Cenário III temos um crescimento de tamanho mais rápido que nos outros dois cenários. A razão dos declives das rectas permite concluir que o tamanho da

estrutura de dados no Cenário III é superior em 1,42 vezes ao Cenário II e que por sua vez, é superior em 1,77 vezes ao Cenário I. No caso do índice de Solr, do lado direito da Figura 5.5 observa-se um comportamento bastante semelhante ao gráfico de Oracle. Por outro lado, concluímos que o efeito anteriormente notado na Figura 5.4 se deve ao índice de Solr. Note-se que este efeito, ocorre precisamente ao mesmo nível nos três cenários. Para o justificar, temos de recorrer a um parâmetro de configuração da instalação de Solr: *MergeFactor*. Este factor determina qual o número de segmentos de igual tamanho que serão construídos antes de os unir num único segmento [Sol11]. Se este factor tiver um valor grande, isso traduz-se em menos uniões e consequente melhor tempo de indexação, mas com mais ficheiros de índice, logo com pior tempo de pesquisa. No caso inverso, as consequências ocorrem inversamente. A união dos segmentos é a provocadora da optimização do índice e por conseguinte, do efeito de "degraus". Ao optimizar o índice, também ocorre a consequência inversa, embora numa escala menor, de aumento da latência de inserção dos *batches*. Para confirmar esta tendência podemos verificar nas Figuras 5.1, 5.2 e 5.3 o efeito de "degraus" ainda que mais ligeiro nas curvas referentes a Solr. Neste trabalho, a este factor foi atribuído o valor 25, que é o valor por omissão, e que é considerado um valor elevado.




(a) Oracle

(b) Solr

Figura 5.5: Dimensão dos índices criados em ambas as plataformas à medida que o número de *batches* inseridos aumenta, nos três cenários.

Com este teste concluímos que Solr tem melhor desempenho que Oracle quer a nível de espaço, ainda que não seja muito significativo, quer a nível temporal. Neste último caso, Oracle sofre uma grande penalização quer com o aumento do número de campos a indexar quer com o aumento da quantidade de informação já armazenada na base de

dados. Já Solr tem um comportamento muito previsível e muito pouco influenciado, com aumentos muito ligeiros nas latências, quer com a variação de cenário quer com a quantidade de informação já armazenada.

### 5.1.2 Teste dos Registos

O Teste dos Registos pretende apurar o comportamento dos sistemas ao receber grandes quantidades de informação.

Iniciando o teste pelo Cenário I, que está exposto na Figura 5.6, verifica-se a mesma situação do Teste dos Batches no mesmo cenário, em que Oracle tem melhor performance que Solr. Cada ponto de cada curva representa a razão do tempo de inserção de um *batch* pelo número de registos contidos no *batch*. Ambas as plataformas têm comportamentos bastante regulares sem perda da performance com o aumento do número de registos por *batch*. Excepção feita à parte final do teste em Solr, onde o seu comportamento se torna bastante instável. A partir da marca dos 130 mil registos, cada *batch* tem um tamanho superior a 60 MBytes, o que no caso de Solr activa a necessidade de utilização de memória virtual. Recordando à Secção 4.2, pode-se relembrar que a máquina virtual onde correm os testes conta apenas com 2 GBytes de memória RAM, no entanto, através de medições frequentes conclui-se que conta apenas com metade desse espaço disponível para a plataforma. Assim, conclui-se que Solr faz uma utilização mais expressiva da memória RAM, o que cria a necessidade de utilização de memória virtual. Oracle não chega a utilizar memória virtual.

Nos Cenários II e III, Figuras A.1 e A.2 respectivamente, a tendência mantém-se: aumentando o número de campos a indexar, Solr necessita mais cedo da memória virtual e Oracle acaba por não fazer uso da memória virtual. Excluindo a instabilidade de Solr na parte final dos testes, ambas as plataformas seguem uma tendência constante.

Analisando o espaço ocupado pelos índices criados, que pode ser consultado no Figura 5.7 é seguida a mesma tendência que se verifica no Teste dos Batches, à excepção de neste caso Solr ter agora um capacidade de optimização claramente superior ao Oracle. Mais especificamente nos Cenários II e III, Solr constrói um índice que ocupa entre 40% a 70% do espaço ocupado pelo índice do sistema homólogo.

O Teste dos Registos mostra-nos uma supremacia do Oracle relativamente ao Solr em todos os cenários, a nível de latências de inserção dos dados. A nível de espaço, temos uma performance significativamente melhor do Solr.

### 5.1.3 Teste dos Campos de Texto

O Teste dos Campos de Texto pretende verificar as tendências das plataformas ao inserir registos em estruturas com um número de campos de texto variável.

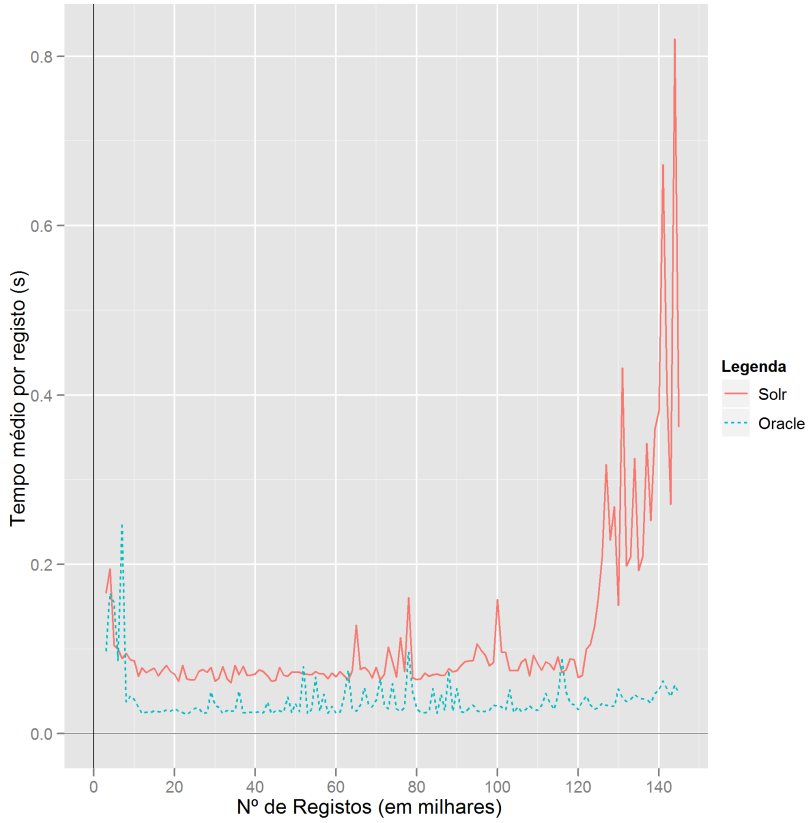



Figura 5.6: Latência média de inserção de cada registo à medida que aumenta o número de registos em cada *batch*, referente ao Cenário I.

Este teste tem bastantes semelhanças com os dois primeiros testes do caso de uso "inserir". No Cenário I (Figura 5.8), onde é indexado um único campo de texto, cada ponto de cada curva representa a latência total de inserção de um *batch* com 250 registos e com um número de campos variável. Oracle inicia o teste com melhor desempenho, no entanto, a partir do patamar dos 120 campos de texto as latências do Oracle disparam. A cada 100 campos de texto adicionados à estrutura de dados, a latência de inserção de cada *batch* em Oracle aumenta aproximadamente um segundo. Já Solr segue uma tendência constante em torno do nível dos 800 milisegundos, mantendo-se insensível ao aumento do número de campos de texto na estrutura.

Se atentarmos no eixo vertical das latências, podemos concluir que as latências apresentadas neste teste são bastante superiores às latências dos Testes dos Batches e dos Registos. A justificação consiste na forma como foram construídas as curvas de cada plataforma. Cada curva foi criada através do registo da latência total de inserção de um *batch*, onde o número de campos de texto varia e nos dois primeiros testes são registados o tempo médio por *batch* e o tempo médio por registo, o que se traduz em latências mais baixas que as apresentadas nos gráficos do corrente teste.

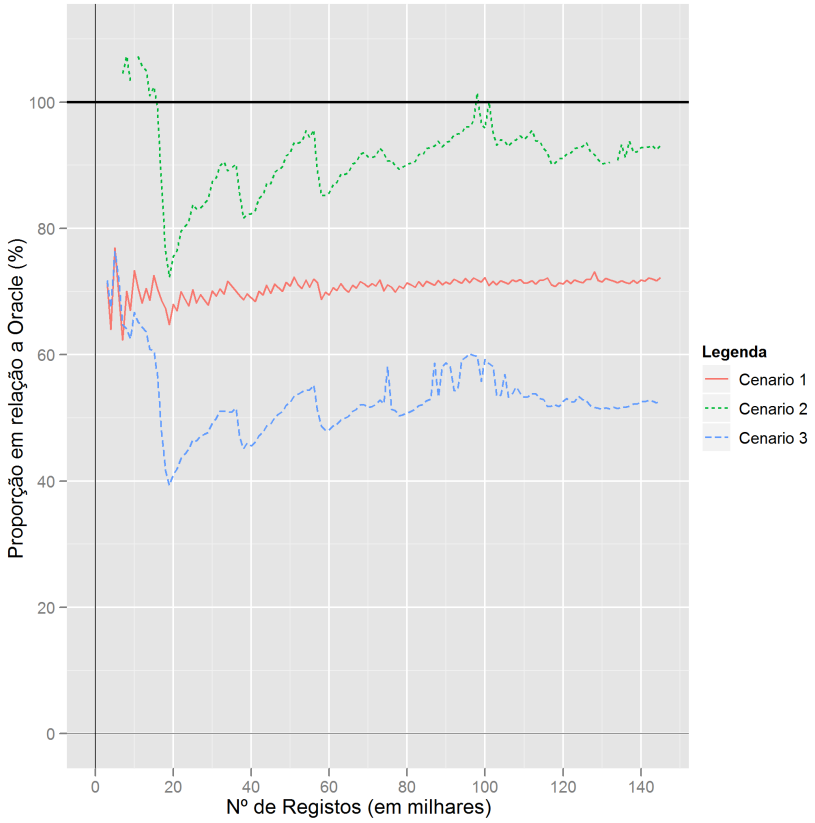



Figura 5.7: Comparação das estruturas de dados geradas em Solr, comparativamente a Oracle nos três cenários referentes ao Teste dos Registos.

Nos Cenários II e III, a que dizem respeito as Figuras A.1 e A.2 respectivamente, as tendências acentuam-se mais. As latências em Oracle aumentam relativamente ao Cenário I aproximadamente cinco e dez vezes, para os Cenários II e III respectivamente. Solr mantém uma curva com uma inclinação muito baixa, confirmando toda a sua apetência para indexar texto, não chegando a ultrapassar o patamar dos 2 segundos em nenhuma execução do teste, em qualquer dos cenários. A nível de espaço, Solr obtém um resultado notável. Na Figura 5.9 constata-se que o Solr cria uma estrutura com metade do espaço ocupado pela estrutura do Oracle, em grande parte do teste. Esta capacidade de optimização de espaço do Solr atinge ainda valores mais destacados nos cenários onde são indexados 10 e 20 campos de texto, ao criar uma estrutura com um tamanho na ordem dos 20% do tamanho da plataforma homóloga.

Neste teste, Solr apresenta-se bastante preparado para cenários muito exigentes a nível de indexação e mostra uma supremacia inédita em relação ao Oracle. Tanto a nível de tempos de inserção, como a nível de espaço ocupado Oracle tem resultados muito inferiores e mostra que a sua configuração necessita claramente de ser adaptada ao domínio do problema.

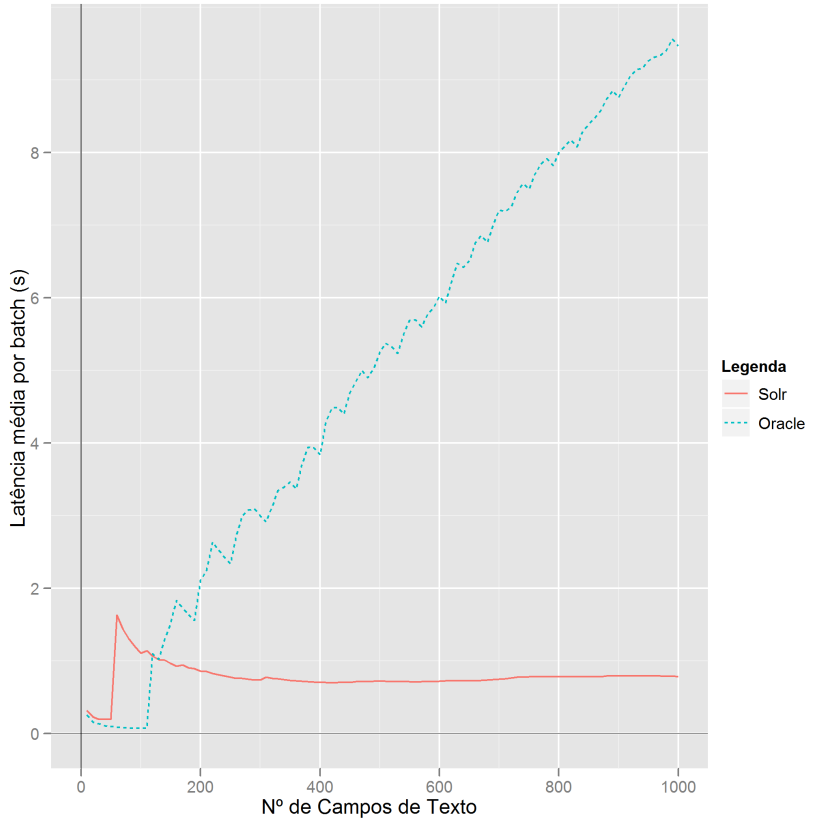



Figura 5.8: Latência média de inserção de cada *batch* à medida que aumenta o número de campos de texto da base de dados, referente ao Cenário I.

### 5.1.4 Teste dos Campos Indexados

À partida, pelas conclusões obtidas a partir dos três primeiros testes, prevê-se que a plataforma Solr esteja melhor preparada que Oracle para indexar um grande conjunto de campos de texto.

O Teste dos Campos Indexados não foi desenhado para ser executado nos três cenários já referidos, uma vez que o objectivo do teste passa pela variação do número de campos de texto a serem indexados. Assim, em todas as execuções deste teste, os sistemas constroem uma estrutura de dados com um conjunto máximo de campos de texto e gradualmente aumentam o número de campos que indexam.

A partir da Figura 5.10, observa-se que Oracle e Solr têm um desempenho muito parecido com o desempenho no Teste dos Campos de Texto, que pode ser consultado a partir das Figuras 5.8, A.3 e A.4. Ou seja, as latências referentes a Oracle crescem a um ritmo muito maior que em Solr. Solr continua com uma latência de inserção de cada *batch* inferior a 2 segundos ao longo de todo o teste. Quanto ao Oracle, verifica-se um desempenho mais irregular quando comparado com o Teste dos Campos de Texto, mas

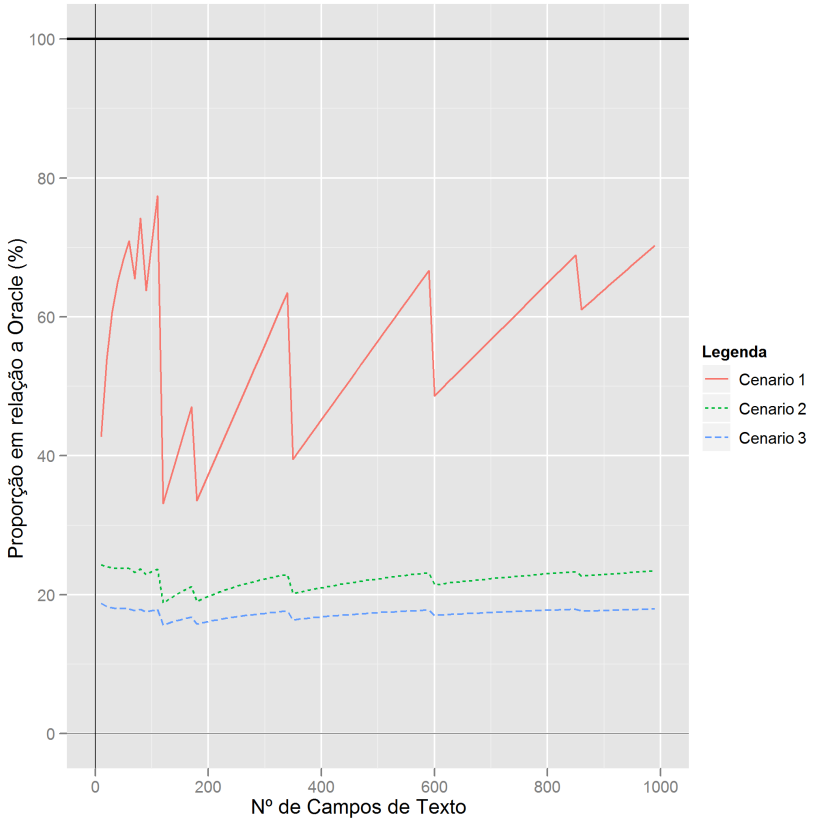



Figura 5.9: Comparação das estruturas de dados geradas em Solr, comparativamente a Oracle nos três cenários referentes ao Teste dos Campos de Texto.

mesmo assim, com um grau de degradação das latências semelhante.

Analisando este teste em termos de espaço, vemos na Figura 5.11 uma curva muito curiosa pela sua regularidade, sem o efeito de "degraus" visível nos Testes dos Batches e dos Registos. A quantidade de dados a inserir em qualquer execução deste teste é a mesma (aproximadamente 4,72 MBytes) e é uma quantidade muito pequena quando comparada com a quantidade de informação a inserir nos dois primeiros testes, onde nos piores casos chega respectivamente a 300 Mbytes e a 90Mbytes. Deste modo, o número de segmentos criados em qualquer execução do teste no índice em Solr não chega ao factor de união (*MergeFactor*) e por conseguinte, não dá origem ao efeito de "degraus" criado por Solr nos primeiros dois testes. Por outro lado, o índice criado em Oracle tem um crescimento muito estável, ao contrário do que acontece, por exemplo no Teste dos Campos de Texto e que pode ser visto na Figura 5.9. Nesta situação, a dimensão do índice é bastante baixa e de uma execução para a seguinte, basta praticamente o espaço livre nos segmentos alocados pela estrutura para indexar um novo *batch*. Assim, Oracle também não gera o efeito de "degraus" e isto permite que se note uma hipérbole perfeita a olho nu, que pode ser vista na Figura 5.11.

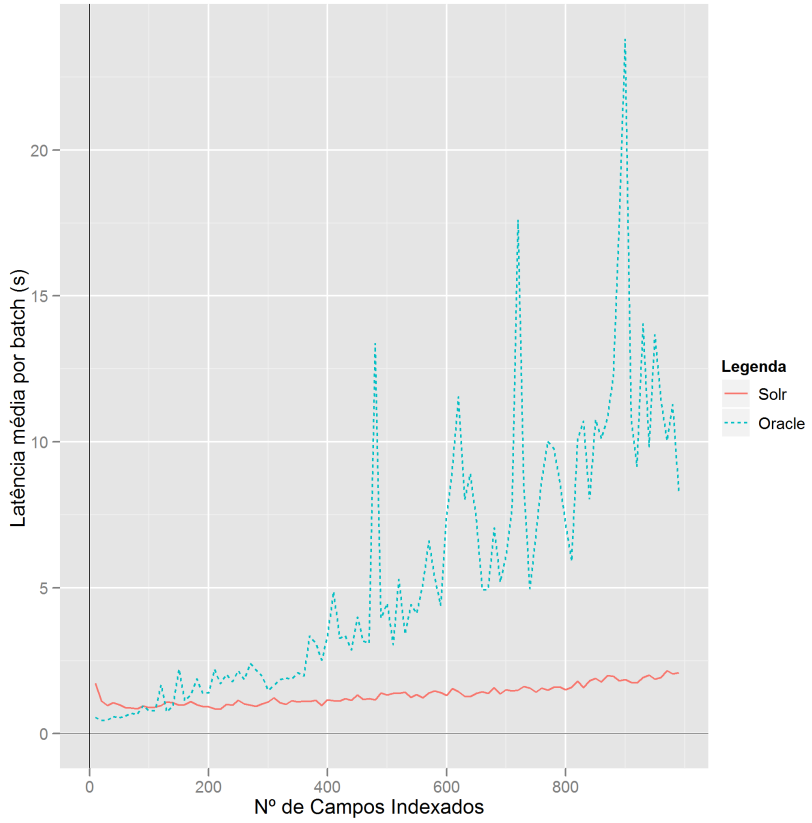



Figura 5.10: Latência média de inserção de cada *batch* à medida que aumenta o número de campos a indexar, referente ao Teste dos Campos Indexados.

No cômputo final, as conclusões a retirar deste teste são em tudo semelhantes ao Teste dos Campos de Texto. Estes dois testes põem à prova a capacidade de indexação de grandes quantidades de informação de ambas as plataformas e Solr comprova a sua vocação apresentando um excelente desempenho a todos os níveis. Oracle revela uma necessidade clara de refinamento da sua configuração para produzir resultados ao nível da plataforma homóloga.

### 5.1.5 Teste dos Campos de Tipo Inteiro

O Teste dos Campos do Tipo Inteiro é idêntico ao Teste dos Campos de Texto, à excepção da variável utilizada, que ao invés do número de campos de texto, é o número de campos do tipo inteiros. As semelhanças prolongam-se ainda aos resultados obtidos.

O Cenário I deste teste (Figura 5.12), tem um aspecto idêntico ao mesmo cenário do Teste dos Campos de Texto (Figura 5.8), embora numa escala temporal bastante menor porque naturalmente o peso da indexação de um número inteiro é inferior ao de um campo

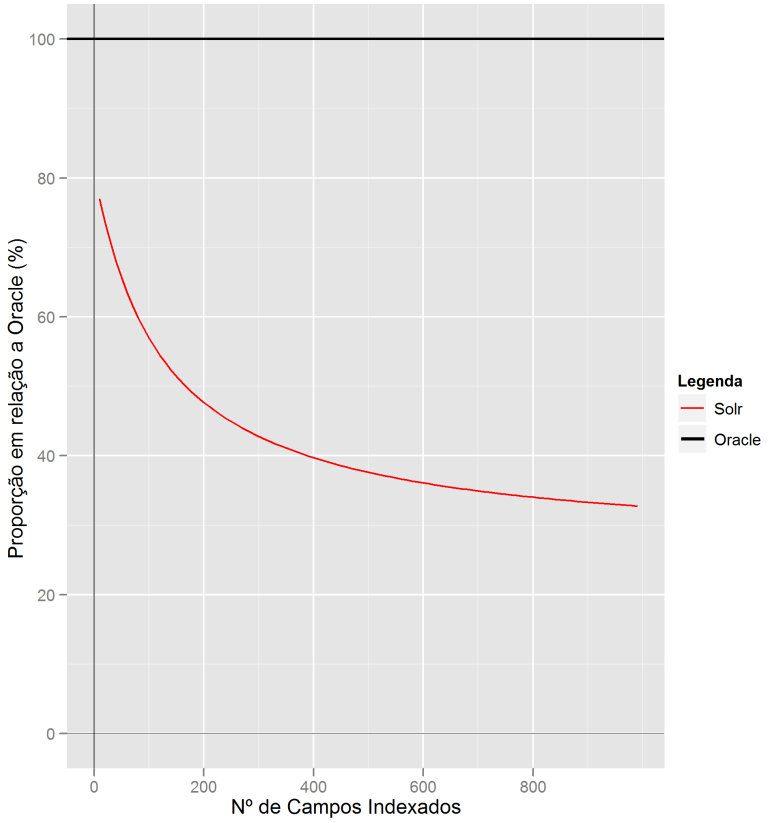



Figura 5.11: Comparação das estruturas de dados geradas em Solr e em Oracle, referentes ao Teste dos Campos Indexados.

de texto. Oracle apresenta uma curva consideravelmente mais inclinada e mais irregular que Solr.

Os Cenários II e III deste teste, onde são indexados respectivamente 10 e 20 campos de texto, estão representados na Figura 5.13. O presente teste tem a particularidade do factor a que está associado, o número de campos do tipo numérico, ser totalmente independente da relação entre os três cenários desenhados neste trabalho. Por exemplo, nos Testes dos Batches e dos Registos, a quantidade de informação a inserir na base de dados varia, logo a variação do cenário implica mais ou menos carga na plataforma, consoante se aumenta ou diminui o número de campos a indexar. Neste teste, não existe uma relação deste género, e por conseguinte, à variação dos cenários corresponde apenas um deslocamento na vertical entre gráficos, como é bem visível na Figura 5.13.

Comparando as plataformas em termos de espaço, temos uma situação inédita em que Oracle tem no global uma performance superior a Solr como se pode ver na Figura 5.14. Apenas é apresentado o Cenário I uma vez que à imagem da análise feita a nível de tempo de indexação, as curvas associadas aos outros dois cenários apenas estão deslocadas na vertical e não acrescentam nenhuma tendência pertinente. A análise do tamanho do índice

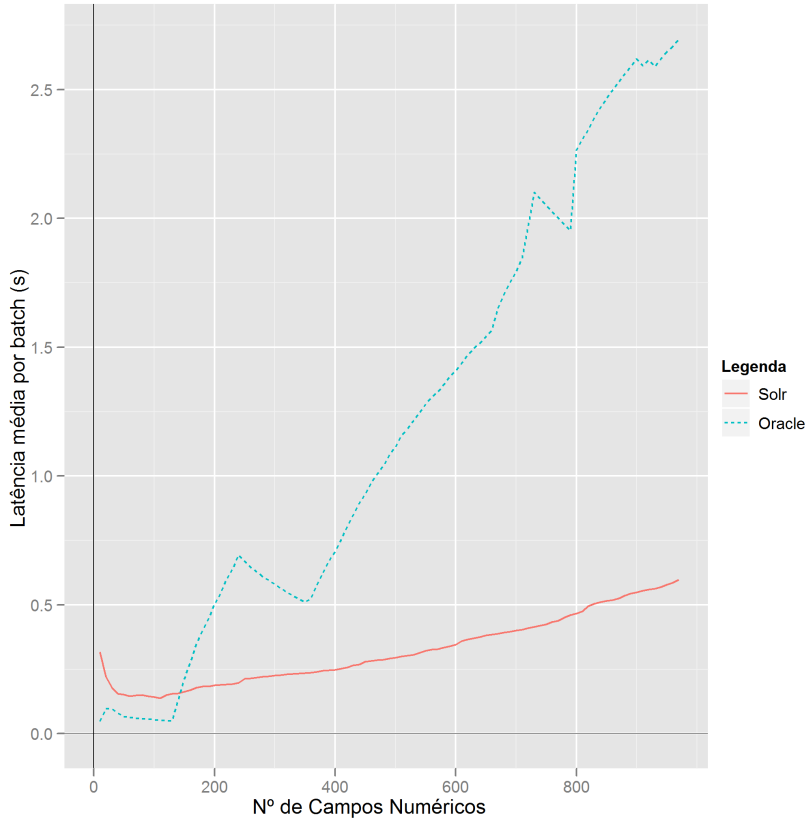



Figura 5.12: Latência média de inserção de cada *batch* à medida que aumenta o número campos de tipo inteiro na estrutura de dados, referente ao Cenário I.

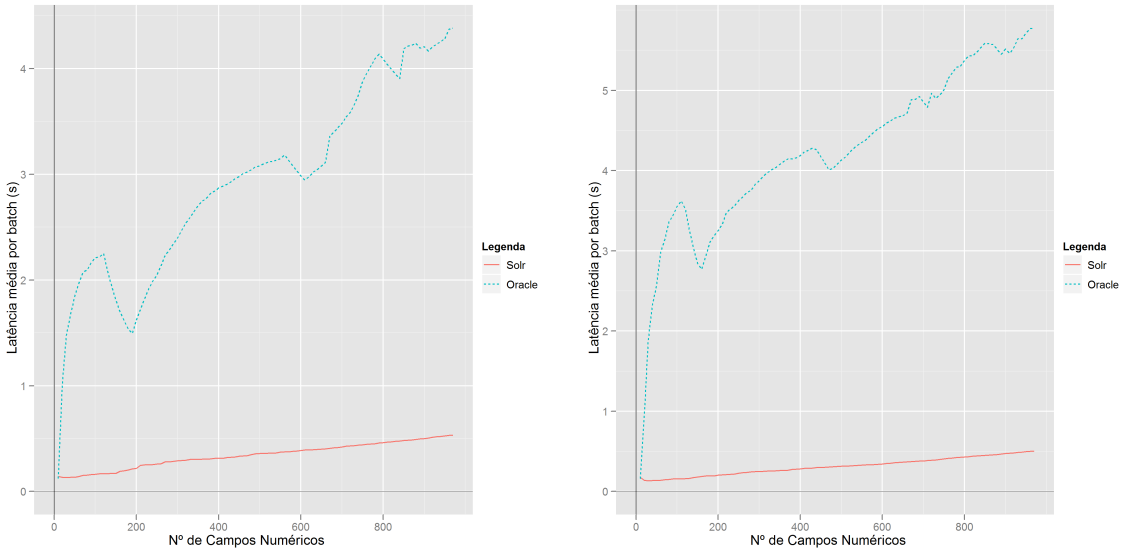
criado pelas duas plataformas está bastante condicionada pelo tamanho da informação a indexar e o tamanho dos segmentos utilizados em cada plataforma. A Figura 5.15 permite comparar o tamanho dos índices das duas plataformas. Como se pode perceber pelo gráfico do Oracle, ocorre novamente a formação do efeito de "degraus" o que tem como consequência a irregularidade da Figura 5.14.

Neste teste, Solr mostra-se novamente superior ao Oracle a nível de tempo de indexação, no entanto, Oracle apresenta melhor eficiência a nível de espaço.

### 5.1.6 Teste dos Índices de Texto

O Teste dos Índices de Texto consistirá numa repetição do Teste dos Registos, no entanto, fazendo variar o tipo de índice que cada plataforma utilizará. Assim, o objectivo deste teste é comparar o comportamento dos sistemas a acumular informação, quer ao indexar um campo por inteiro, quer ao indexar o campo por palavras.

Uma vez que este teste será analisado de uma forma mais exaustiva que os testes anteriores, apenas foi concebido para o cenário intermédio, que se interpreta como o mais




(a) Cenário II

(b) Cenário III

Figura 5.13: Latência média de inserção de cada *batch* à medida que aumenta o número campos do tipo inteiro, referente aos Cenários II e III.

justo para as plataformas, onde são indexados 10 campos de texto dos 20 armazenados.

Note-se que alguns dos gráficos que são apresentados neste teste comparam índices que indexam os dados com vista a fins diferentes, podendo permitir ou não a pesquisa por palavras-chave, o que não é uma comparação justa. No entanto, o objectivo desta comparação é ter uma noção do custo da possibilidade de poder pesquisar um campo por palavras, ou a obrigatoriedade de pesquisá-lo por inteiro (pesquisar a frase inteira).

Relativamente ao Solr, não é possível escolher o tipo de índice que utiliza. Apenas é permitida a aplicação um conjunto de filtros ao índice, funcionalidade comum a Oracle. No que diz respeito a este teste, foi apenas aplicado um filtro que é comum a Oracle, o filtro de divisão por espaços (*whitespace tokenizer*) que permite a pesquisa por palavras. Na Figura 5.16 temos uma comparação das latências de inserção de blocos de 250 registos, à medida que a informação acumula na base de dados, utilizando um índice com e sem filtro de divisão por espaços. O gráfico gerado tem um aspecto curioso: Solr tem um desempenho praticamente idêntico a indexar palavras e a indexar frases por inteiro.

Na Figura 5.17 é feita a mesma análise do parágrafo anterior mas relativamente à plataforma Oracle. Neste caso, surgem diferenças evidentes entre os vários índices. Os índices CTXCAT e CONTEXT pertencentes ao módulo Oracle TEXT, têm um desempenho mais fraco a indexar do que os índices que não fazem a divisão por espaços. Em particular, o índice CTXCAT que é optimizado para trabalhar com quantidades pequenas de texto, que é o caso deste trabalho (cada frase inserida tem um tamanho inferior a 60 caracteres), tem um desempenho curiosamente mais fraco, aproximadamente cinco vezes

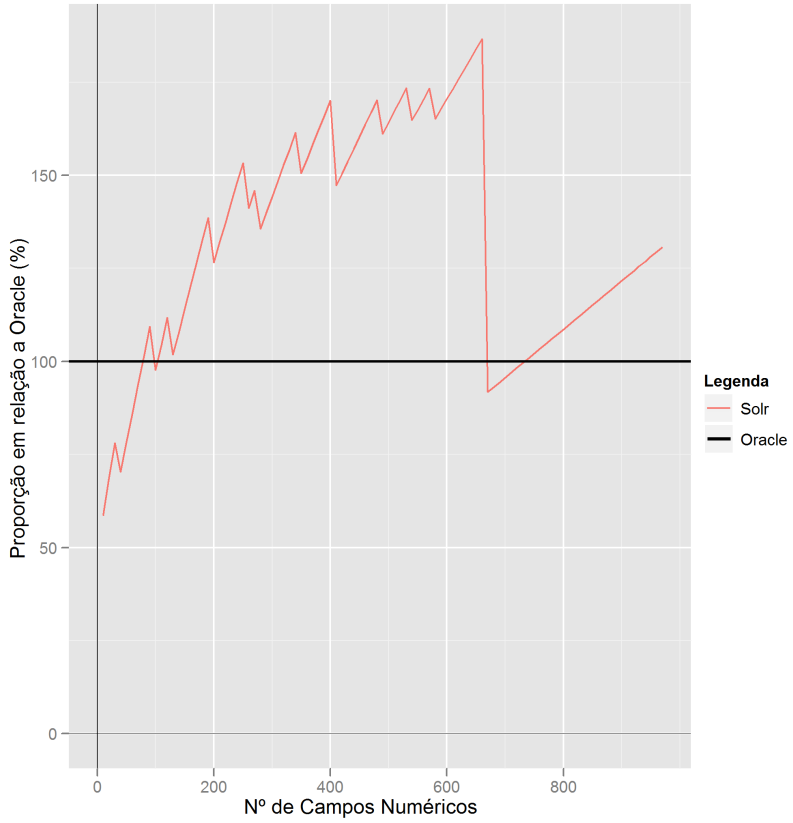



Figura 5.14: Comparação da estrutura de dados gerada em Solr, relativamente a Oracle referente ao Cenário I do Teste dos Campos do Tipo Inteiro.

pior que o índice CONTEXT, vocacionado para trabalhar com grandes quantidades de texto. Relativamente aos índices que não fazem a divisão do campo por espaços, B-Tree e Bitmap, temos de uma forma natural um melhor desempenho do índice B-Tree que é mais adequado a atributos com maior cardinalidade do que o índice Bitmap. Em Oracle é evidente, na parte da indexação, a implicação de um custo elevado entre a indexação de frases ou de palavras, visível pelo desfasamento das curvas dos índices CTXCAT e CONTEXT relativamente às curvas referentes aos índices B-Tree e Bitmap.

Na Figura 5.18 estão postos à prova os apenas índices de texto. No lado esquerdo da figura é feita a análise relativa às latências das consultas. Neste caso, tempos uma superioridade muito clara do Solr desde o início do teste. O índice do Solr tem uma performance superior em média 90 vezes e 12 vezes respectivamente aos índices CTXCAT e CONTEXT. De salientar ainda que o teste feito ao índice CTXCAT terminou bastante mais cedo que os outros índices, uma vez que a taxa de inserção de dados era extremamente lenta (aproximadamente 34 segundos para inserir e indexar 2000 registos) e visualmente já permitia retirar a tendência clara que este tipo de índice apresenta a nível de tempos de indexação.

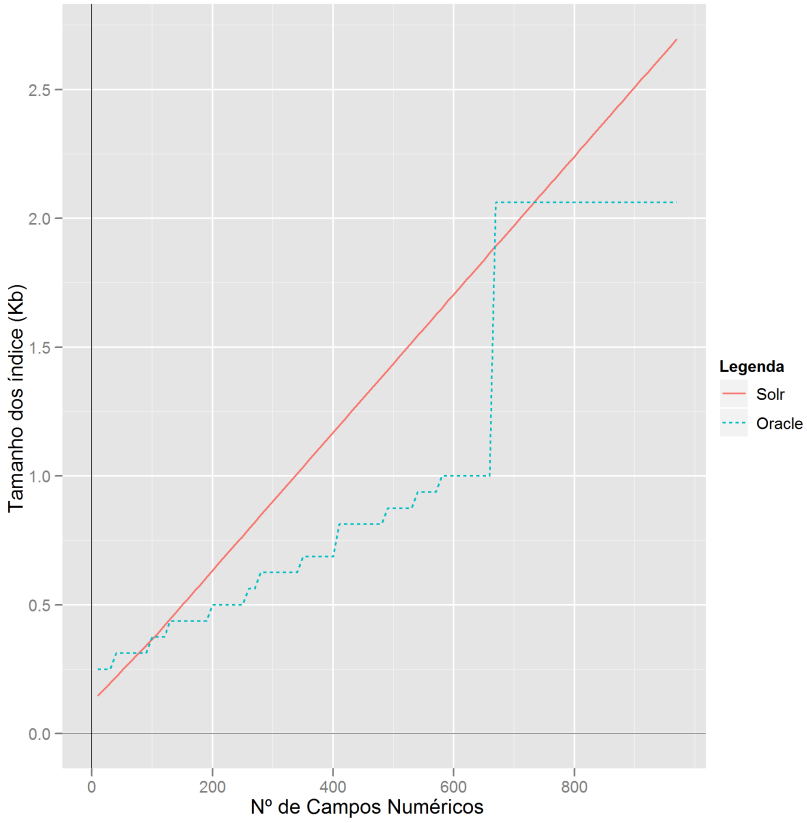



Figura 5.15: Dimensão dos índices criados em ambas as plataformas à medida que o número de campos numéricos aumenta, referente ao Cenário I do Teste dos Campos do Tipo Inteiro.

É óbvio que a configuração das plataformas tem implicações claras nos resultados. Naturalmente que pelos resultados obtidos se percebe novamente que a configuração base de Oracle é ineficiente para ser implementada num ambiente minimamente realista. No caso do Solr, também podiam ter lugar bastantes optimizações, embora a sua configuração base já produza resultados muito satisfatórios. Para efeitos deste teste, se tivessem tido lugar optimizações às plataformas, era muito discutível a fiabilidade dos testes na medida em que as plataformas não iriam ser postas à prova nas mesmas condições.

Para finalizar o teste, segue-se uma análise a nível da dimensão dos índices gerados. No gráfico do lado direito da Figura 5.18 destaca-se o mau desempenho do índice CTX-CAT que também obteve o pior desempenho a nível de latências na indexação. Em ambos os índices de Oracle, nota-se o efeito de "degraus" discutido no teste anterior e no índice CONTEXT onde este efeito é mais visível, é aquele que cria uma estrutura de menor dimensão. De qualquer forma, relativamente ao espaço, a performance dos vários índices não denota diferenças muito significativas.

De um modo geral, neste caso de uso Oracle para acompanhar os resultados de Solr, necessitaria de muitos mais recursos. Oracle tem um melhor desempenho nas situações

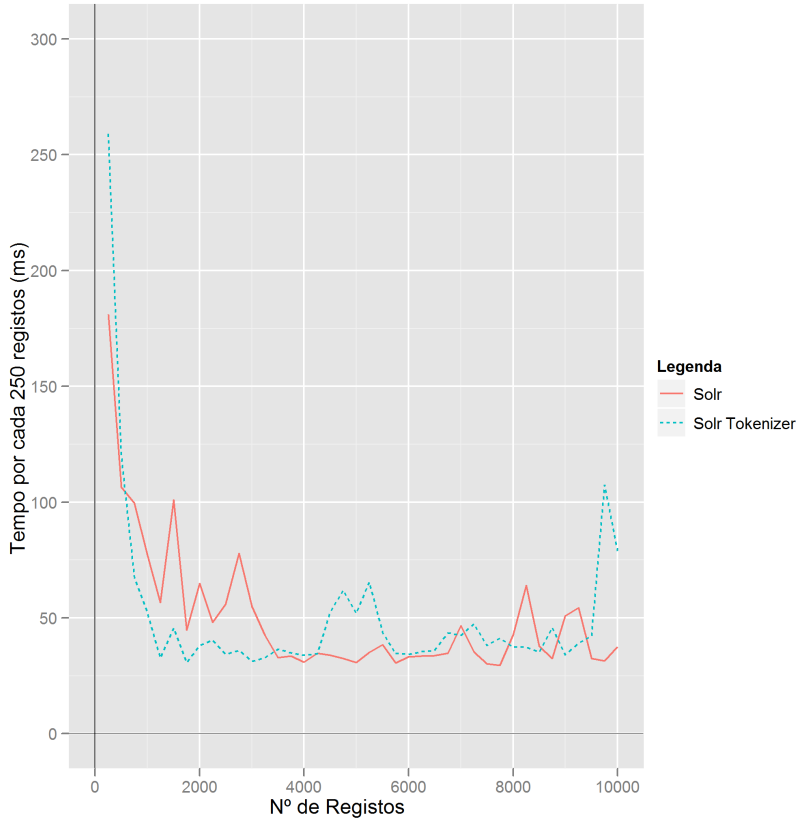



Figura 5.16: Comparação das latências de inserção de dados em Solr utilizando ou não utilizando filtro de espaços.

menos exigentes, mas à medida que aumenta a exigência de cada teste, a performance do Oracle degrada-se a um ritmo muito maior do que a do Solr. Esta situação também é realçada com a variação dos cenários. Em qualquer teste, no cenário onde são indexados todos os campos de texto, Oracle tem um desempenho muito pobre. A nível de espaço ocupado pelos índices, Solr tem resultados ligeiramente melhores. Oracle é um sistema com muitas funcionalidades e com provas dadas, no entanto, para obter resultados do nível do Solr, é necessário refinar bastante a sua configuração e adaptá-la ao domínio do problema, o que não está previsto neste trabalho, tal como foi referido no Capítulo 4. Solr para além de demonstrar um grande desempenho na sua configuração base, ainda gera gráficos com tendências muito claras, o que permite prever com um maior grau de segurança o seu comportamento se os testes continuassem a escalar.

## 5.2 Caso de uso “pesquisar”

O caso de uso “pesquisar” está dividido em seis testes diferentes. Nos primeiros cinco são utilizados os índices por omissão de cada plataforma e no último teste é feita uma

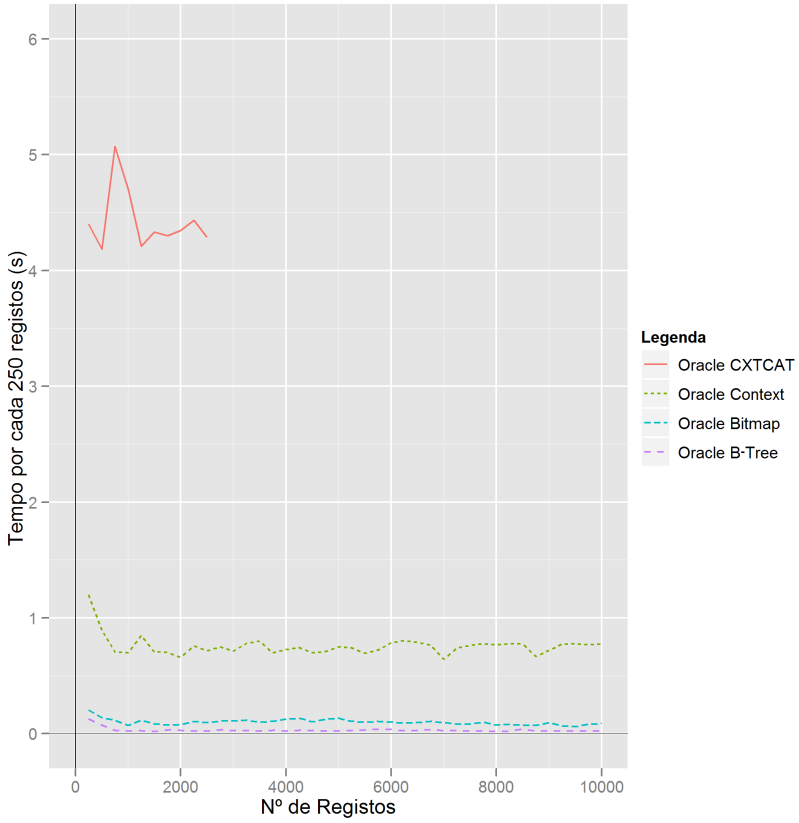


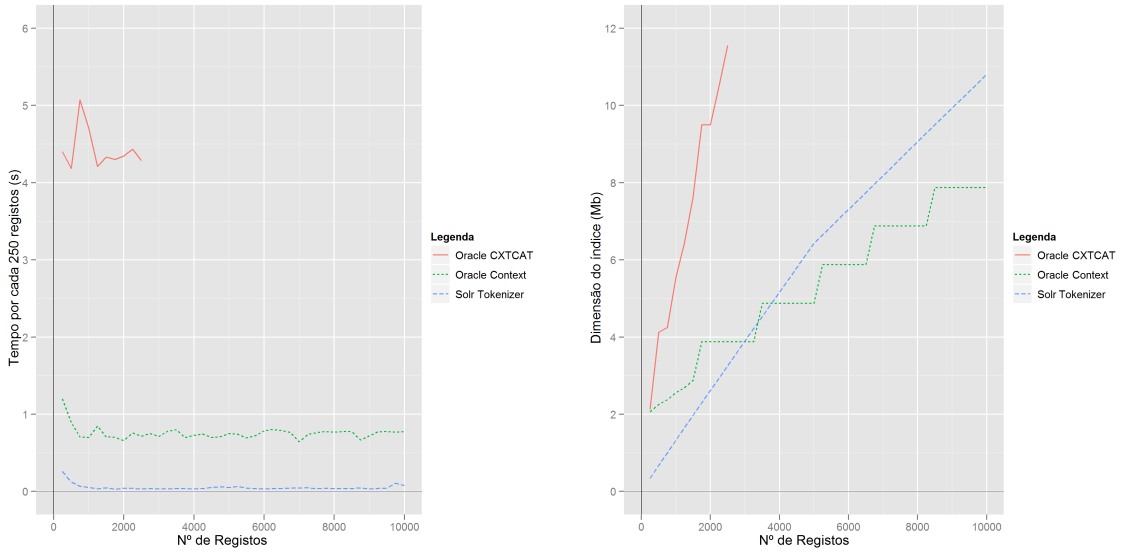

Figura 5.17: Comparação das latências de inserção de dados em Oracle utilizando índices que fazem a divisão por espaços (CTXCAT e CONTEXT) e índices que não fazem a mesma divisão (Bitmap e B-Tree).

repetição do Teste dos Registos, mas utilizando índices que façam a divisão de texto em palavras.

Globalmente, ao longo dos testes deste caso de uso os gráficos gerados não apresentam tendências tão claras como nos outros casos de uso, o que se deve em parte ao facto de neste caso de uso as plataformas terem mais carga e concorrência nas transacções. Pode-se ainda realçar que a diferença na grandeza medida também contribui para que os gráficos sejam mais irregulares. Ou seja, como normalmente as latências das consultas são bastante inferiores às latências das operações de inserção, então um pequeno desvio de uma latência numa escala de tempo em milisegundos tem um impacto maior no aspecto do gráfico do que um outro desvio numa escala de tempo em segundos.

### 5.2.1 Teste dos Registos

Neste teste pretende-se averiguar como variam as latências das consultas às bases de dados, com a quantidade de informação que estas armazenam. A Figura 5.19 representa os gráficos gerados neste testes relativos aos Cenários I e II. No lado esquerdo da figura,




(a) Análise de tempo

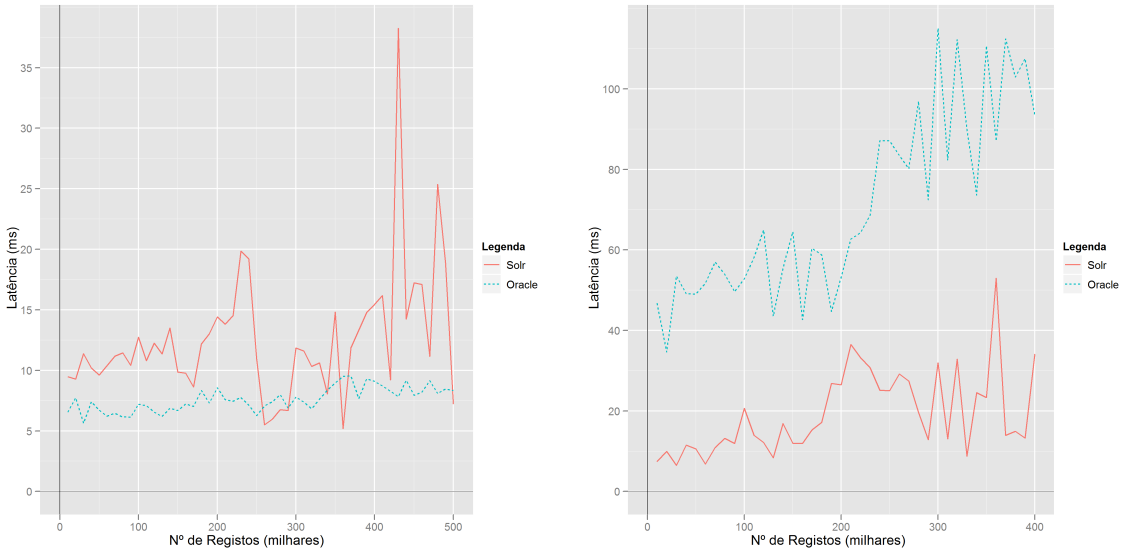
(b) Análise de espaço

Figura 5.18: Análise das latências e das dimensões dos índices em Oracle e em Solr utilizando índices que fazem a divisão por espaços.

o gráfico diz respeito ao cenário onde é apenas indexado um único campo de texto e, no lado direito, o gráfico é referente ao cenário onde são indexados 10 campos de texto.

A partir da Figura 5.19 confirma-se uma tendência previsível, tendo em conta as conclusões obtidas no caso de uso "inserir". Indexando e pesquisando sobre um único campo de texto, temos um desempenho semelhante em ambas as plataformas, com as latências a variar em grande parte no intervalo [5 ; 15] milisegundos. No caso do Oracle percebe-se uma ligeira perda de performance com o aumento do número de registos, já em Solr a amplitude de variação das latências médias é maior. De um modo geral, ainda que não muito significativamente, Oracle tem resultados superiores a Solr. No gráfico do lado direito, onde as plataformas indexam e pesquisam 10 campos de texto, as latências em Oracle aumentam bastante e ainda é notória uma degradação da sua performance acompanhada pelo acumular de registos na base de dados, aproximadamente a partir dos 220 mil registos. Em Solr, a tendência mantém-se próxima do cenário anterior, sem grande definição de ganho ou perda de performance com o acumular de registos na estrutura, no entanto, na parte final do teste Solr apresenta uma amplitude média de latências superior à amplitude das latências iniciais.

Passando ao Cenário III, onde são indexados e pesquisados 20 campos de texto, este pode ser visto como uma extensão do Cenário II. Solr mantém o mesmo comportamento, no entanto, as latências em Oracle têm uma tendência claramente crescente e aumentam para valores superiores a 100 milisegundos, o que é aproximadamente o dobro das latências do Solr, ao longo de todo o teste, como pode ser visto no Gráfico A.5.




(a) Cenário I

(b) Cenário II

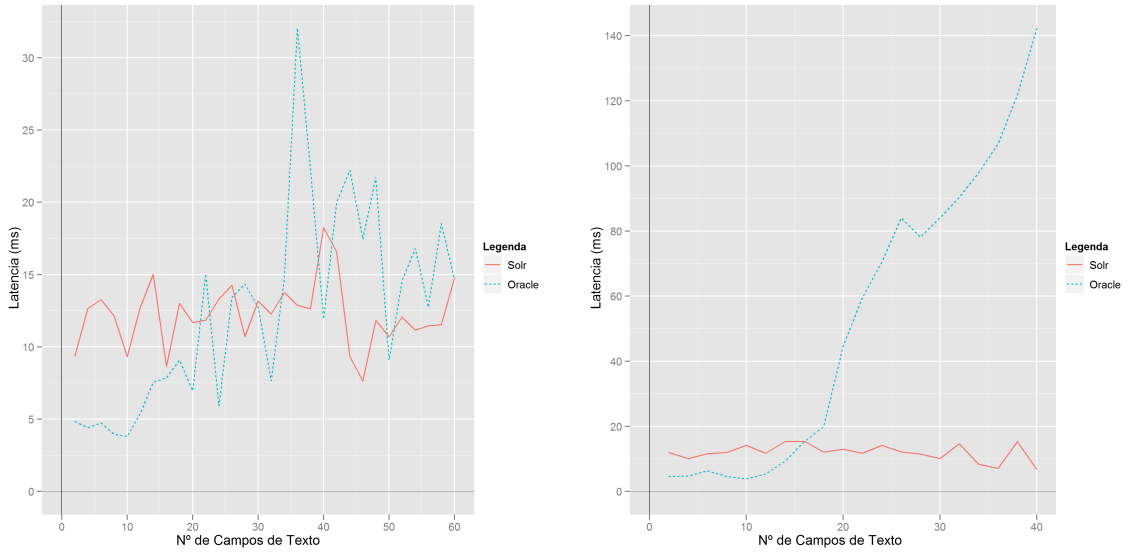
Figura 5.19: Latência média de cada consulta à medida que as plataformas acumulam informação, referentes aos Cenários I e II.

### 5.2.2 Teste dos Campos de Texto

Com este teste pretende-se aferir a forma como variam os tempos de resposta das plataformas à medida que se aumenta o número de campos de texto da base de dados.

Os resultados dos Cenários I e II do presente teste estão descritos visualmente na Figura 5.20. Visualiza-se de novo o mesmo desfasamento entre os dois cenários. No Cenário I, no lado esquerdo da figura, as plataformas têm um desempenho semelhante e, ao invés do que acontece no teste anterior, o gráfico mais irregular pertence ao Oracle. Solr aparenta não perder o nível de desempenho com o aumento dos atributos da estrutura, mas o inverso acontece com Oracle. Neste último, as latências médias de resposta a uma consulta iniciam a 5 milisegundos e no fim do teste terminam três vezes superiores, na ordem dos 15 milisegundos. Ao passar a indexar e pesquisar metade dos campos de texto da estrutura (lado direito da Figura 5.20) ocorre a clássica explosão no gráfico do Oracle, discutida na Secção 5.1.1. Solr segue a sua tendência estável e aproximadamente constante com latências na ordem dos 12 milisegundos tal como o mostra no Cenário I.

O Cenário III (Figura A.6) não acrescenta nenhuma tendência relevante, apenas confirma ainda mais previamente o crescimento exponencial tão característico do Oracle ao longo de todo este *benchmark*.




(a) Cenário I

(b) Cenário II

Figura 5.20: Latência média de cada consulta à medida que aumenta o número de campos de texto na base de dados, referentes aos Cenários I e II.

### 5.2.3 Teste dos Campos a Pesquisar

No Teste dos Campos de Texto observa-se como se comportam as latências médias de consulta dos sistemas à medida que aumenta o número de campos de texto na base de dados, não necessariamente o número de campos de texto a indexar e pesquisar. No referido teste o conjunto de campos de texto a pesquisar varia apenas quando é alterado o cenário escolhido. Já no Teste dos Campos Indexados, o número de campos de texto da base de dados é fixo (pode ser consultado na Tabela 4.3) e o objectivo do teste é variar gradualmente o número de campos de texto indexados e pesquisáveis.

A Figura 5.21 apresenta-nos um comportamento muito semelhante aos cenários mais exigentes, onde são indexados 10 e 20 campos de texto, do teste anterior (respectivamente Figuras 5.20 e A.6). No início do teste, as plataformas apresentam latências estáveis e semelhantes entre si, no entanto, a partir da marca dos 20 campos indexados as latências em Oracle aumentam constantemente e a grande ritmo. Já em Solr, as latências tornam-se mais instáveis no fim do teste e variam num intervalo com grande amplitude. Neste teste, o aumento das exigências a nível de indexação causa um natural impacto na performance do Solr, quer na irregularidade das latências que apresenta, quer na amplitude de variação das latências.

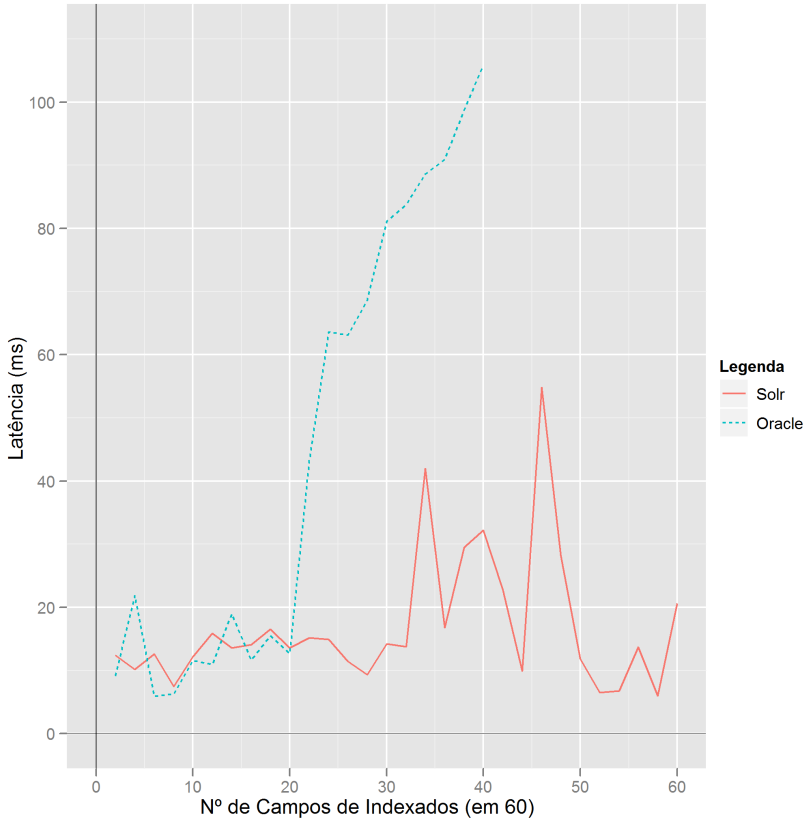
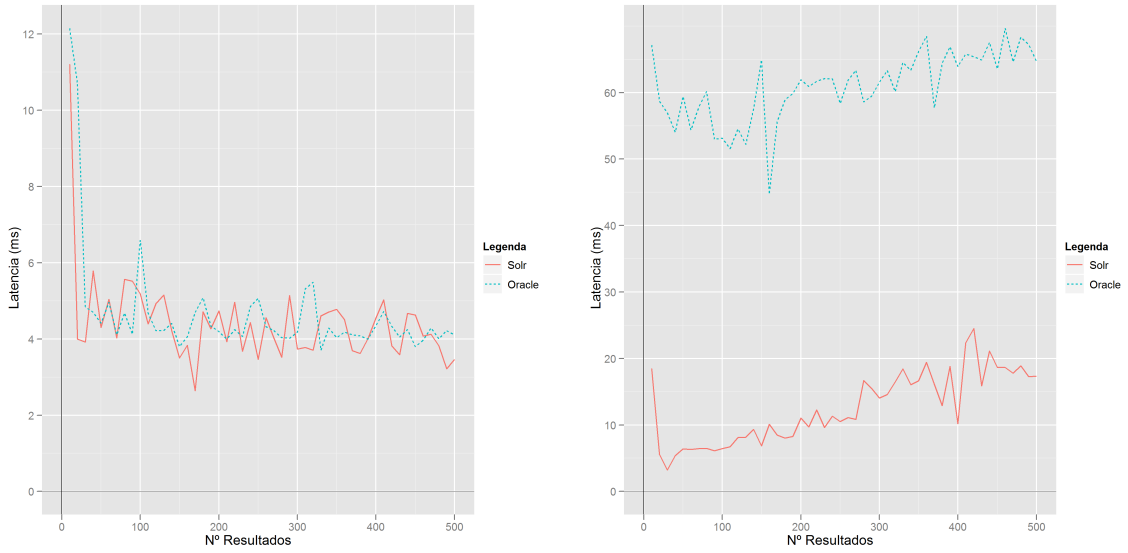



Figura 5.21: Latência média de cada consulta à medida que se aumenta o número de campos de texto indexados e pesquisáveis.

### 5.2.4 Teste dos Resultados

No Teste dos Resultados a estrutura da base de dados é a mesma em todas as execuções. A única variável é o número de resultados máximos que uma consulta pode devolver.

Os resultados dos Cenários I e II do presente teste, podem ser consultados na Figura 5.22, nos lados direito e esquerdo respectivamente. No cenário onde é indexado e pesquisado um único campo ambas as plataformas registam gamas de latências semelhantes e que não se deterioram com o aumento da exigência do teste. Aumentando para 10 campos indexados é fácil verificar tanto em Oracle como em Solr, que o desempenho piora com o aumento do número máximo de resultados devolvidos em cada consulta. No frente-a-frente, Solr apresenta latências cinco vezes inferiores ao Oracle. Do caso menos exigente (um máximo de 10 resultados por consulta) ao caso mais exigente (um máximo de 500 resultados por consulta) as latências aumentam nas duas plataformas em média 10 milisegundos. Isto permite concluir que o factor ”número de resultados devolvidos“ não é problemático para nenhuma das plataformas.




(a) Cenário I

(b) Cenário II

Figura 5.22: Latência média de cada consulta à medida que o número de resultados que cada consulta devolve no máximo aumenta, referentes aos Cenários I e II.

O Cenário III (Figura A.7) não apresenta nenhuma tendência nova, apenas confirma os comportamentos que já foram observados.

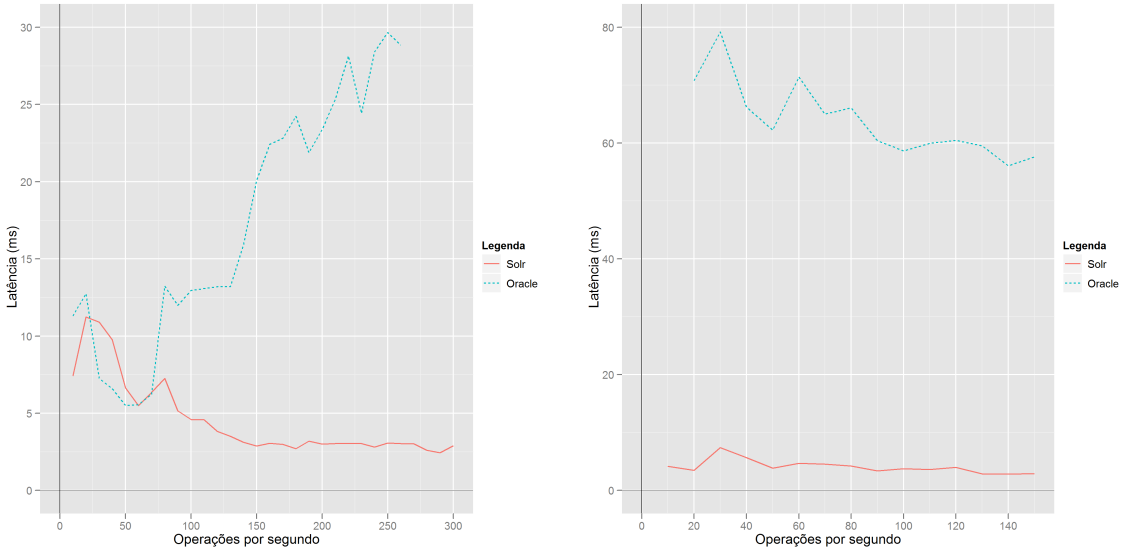
### 5.2.5 Teste da Carga

O Teste da Carga foi desenhado para perceber o comportamento das plataformas à medida que é aumentada a carga sobre as mesmas, sob a forma de operações por segundo a servir.

A estrutura desta secção é semelhante às secções anteriores. Assim temos na Figura 5.23 os resultados dos Cenários I e II do presente teste. Do lado esquerdo temos uma situação que se revelou muito comum ao longo de todos os testes. Oracle inicia o Cenário I com um desempenho superior (ou semelhante) a Solr no entanto, com o aumento da carga no sistema, Oracle ressente-se bastante, ao invés do Solr que tem um desempenho praticamente constante, com uma latência média de aproximadamente 3 milisegundos. A servir 250 operações por segundo, Oracle regista uma latência média dez vezes superior à da plataforma Solr. Do lado direito da Figura 5.23, referente ao cenário da indexação de 10 campos de texto, a latência média para servir uma consulta em Oracle segue mais estável mas aumenta em média 2,5 vezes relativamente ao Cenário I. Em Solr, a latência média aumenta em aproximadamente 2 vezes relativamente ao Cenário I, mas continua com uma capacidade de resposta muito superior à do Oracle.

Um facto curioso que se pode observar na Figura 5.23, no caso específico do Oracle,

é uma perda de performance vincada no Cenário I. No Cenário II, o desempenho da plataforma mantém-se de certa forma inalterado. Não devemos assumir que neste cenário o desempenho de Oracle melhora à medida que aumenta a carga, uma vez que as execuções iniciais podem sofrer influência da compilação *just-in-time*, tal como clarificado na Secção 4.3, mas é justo concluir que a se performance não se degrada com o aumento da carga. Uma outra justificação para haver execuções iniciais com um desempenho pobre perante as execuções seguintes é o facto de alguns resultados intermédios nas plataformas serem armazenados em cache, para no caso de se voltar a processar a mesma transacção, recorrer à cache para imediatamente obter os resultados [Sch].




(a) Cenário I

(b) Cenário II

Figura 5.23: Latência média de cada consulta à medida que a carga aumenta nas plataformas, referentes aos Cenários I e II.

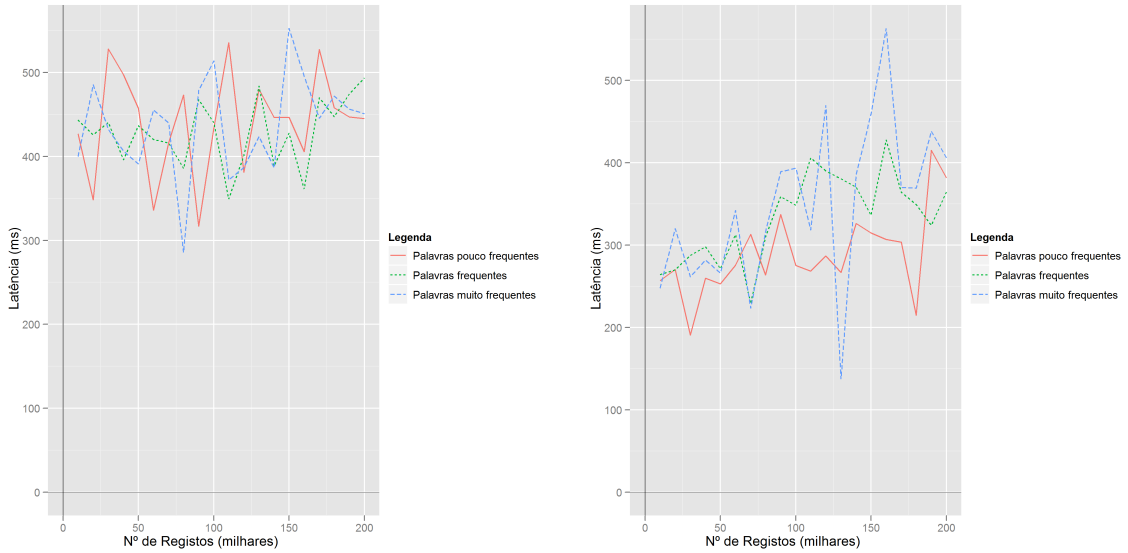
A transição para o Cenário III (Gráfico A.8) é muito semelhante ao Cenário II e não acrescenta nenhuma tendência nova sobre o comportamento das plataformas.

### 5.2.6 Teste dos Índices de Texto

O Teste dos Índices de Texto foi desenhado para avaliar o desempenho dos índices de texto, a nível de latências de pesquisas. Neste teste é variada a quantidade de informação na base de dados e foi repetido pesquisando as palavras de cada um dos grupos criados: "raras", "frequentes" e "muito frequentes" (Secção 4.5).

Na Figura 5.24 estão lado-a-lado as latências de pesquisa dos índices CONTEXT e CTXCAT, do módulo Oracle TEXT. No teste homólogo do caso de uso "inserir" (Secção 5.2.6), apesar do índice CTXCAT ser mais optimizado para trabalhar com pequenas

quantidades de texto, obteve um prior desempenho a indexar. Apesar de tudo, ao pesquisar apresenta latências substancialmente menores que o índice CONTEXT, em média inferiores 100 milisegundos. Por outro lado, o desempenho do índice CONTEXT não se deteriora com a variação da quantidade de informação inserida na base de dados. Já o mesmo não acontece no índice CXTCAT. Se no ínicio do teste, a gama de latências apresentadas é próxima dos 260 milisegundos, no fim do teste, o intervalo de latências apresentado varia entre os 360 e os 410 milisegundos. Atentando aos conjunto de palavras, não se vislumbra nenhuma diferença significativa entre a pesquisa de palavras com mais ou menos frequência, em qualquer dos índices.




(a) Índice CONTEXT

(b) Índice CXTCAT

Figura 5.24: Latência média de cada consulta registada nos índices de texto Oracle, ao pesquisar os vários conjuntos de palavras.

A Figura 5.25 permite ter uma percepção do desempenho do índice de texto do Solr. A tendência deste índice é semelhante ao do índice CONTEXT do Oracle TEXT, no sentido em que as latências não pioram com o aumento da informação inserida na base de dados e novamente, não existe nenhuma evidência de que a variação da frequência das palavras a pesquisar tenha impacto no desempenho do índice.

Na Figura 5.26 estão reunidos os índices de texto do Oracle e do Solr para o conjunto de palavras "raras". A conclusão é clara: Solr tem um desempenho muito superior aos índices de texto do Oracle e entre os índices de texto do Oracle, CTXCAT tem uma performance superior ao índice CONTEXT. As latências registadas nas pesquisas em Oracle são impraticáveis ao longo de todo o teste. Aqui ainda se torna mais claro que a configuração base de Oracle nem em cenários pouco exigentes permite obter resultados satisfatórios, como pode ser visto nas execuções iniciais deste teste.

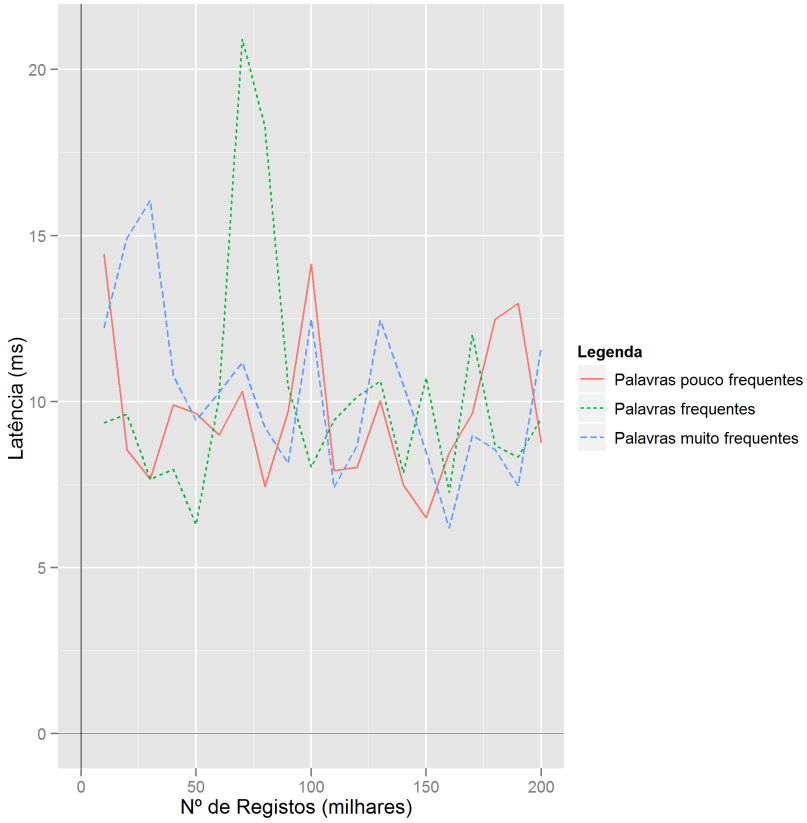



Figura 5.25: Latência média de cada consulta registada no índice de texto do Solr, ao pesquisar os vários conjuntos de palavras.

Globalmente, no caso de uso "pesquisar" retiramos conclusões semelhantes ao caso de uso "inserir". Oracle é novamente muito penalizado com o aumento do número de campos a indexar. No cenário menos exigente, onde é apenas indexado e pesquisado um único campo de texto, o desempenho do Oracle encontra-se ao nível do Solr. No entanto, ao alterar o cenário rapidamente ficam comprometidas as latências das consultas por parte de Oracle.

Ao utilizar índices de texto a análise das latências é ainda mais elucidativa. Para Oracle proporcionar a pesquisa por palavras-chave de um modo eficiente exige-se que a sua configuração seja adaptada especificamente ao ambiente onde está instalado e que sejam disponibilizados mais recursos, uma vez que os tempos de resposta obtidos nas experiências encontram-se na ordem dos 400 milisegundos, que comparam com as latências de 10 milisegundos obtidas por Solr no mesmo teste. Solr, por seu lado, confirma que o seu desempenho sofre muito pouca influência das exigências de indexação.

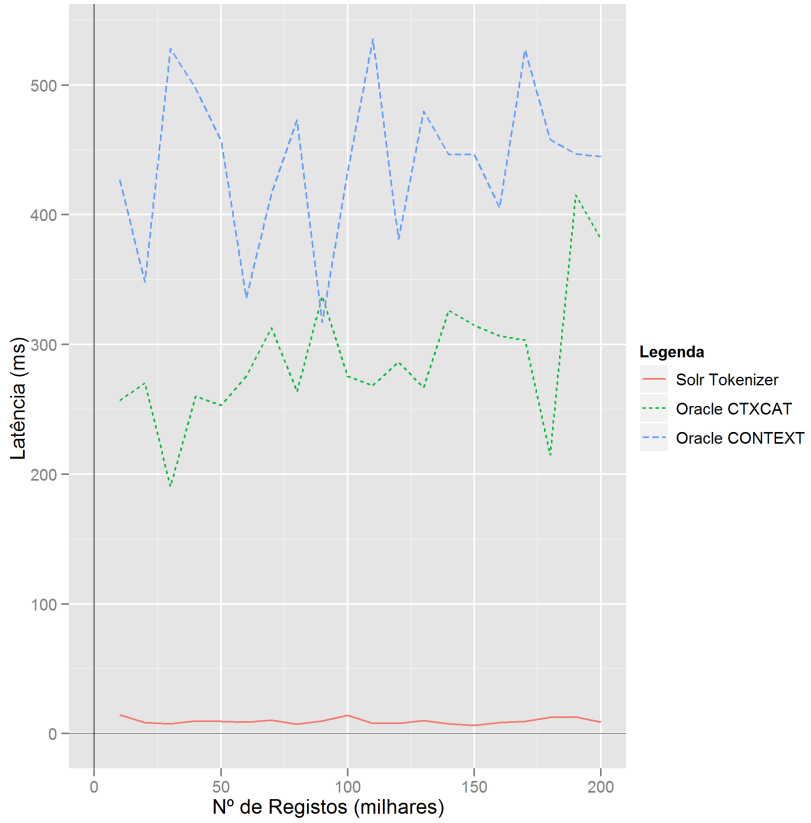



Figura 5.26: Latência média de cada consulta registada nos índices de texto do Oracle e do Solr, ao pesquisar palavras do conjuntos das palavras raras.

## 5.3 Caso de uso "indexar novo campo"

Aquilo que se pretende obter neste caso de uso é a reacção que as plataformas têm à alteração dos esquemas da base de dados. Neste caso em específico, é reproduzida uma operação muito recorrente nos sistemas operados pela PT Inovação: adição de um novo campo de texto, consequente criação de índice e população do campo com informação.

A comparação dos desempenhos das duas plataformas é retratada na Figura 5.27. Neste teste a variável é a quantidade de registos na base de dados, o que claramente não tem influência na performance de Solr, visto apresentar nas execuções mais exigentes do teste latências da mesma gama das execuções menos exigentes. Já em Oracle o desempenho é bastante influenciado com a quantidade de informação inserida na base de dados, uma vez que no fim do teste a gama de latências registada é superior dez vezes à gama de latências no início do teste, que curiosamente é semelhante às latências de Solr ao longo de todo o teste.

Como referido na Secção 4.4.3, neste teste a plataforma Oracle realiza mais operações que Solr. Assim, pretende-se quantificar qual o peso de cada operação na globalidade

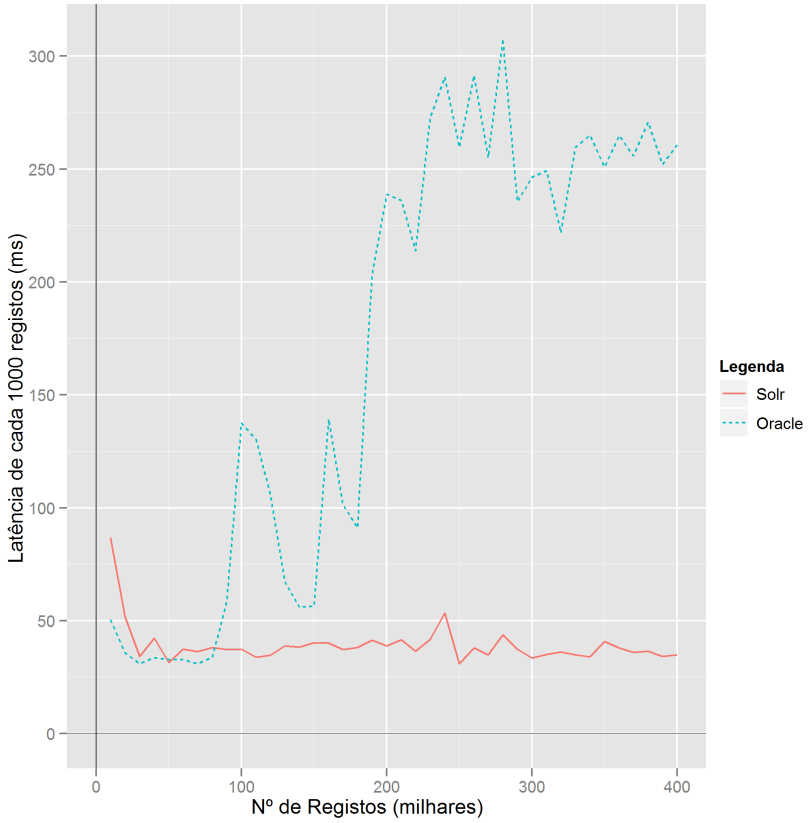



Figura 5.27: Latência de criação de um índice sobre um novo campo medida por cada inserção de 1000 registos.

deste caso de uso. A Figura 5.28 detalha a contribuição de cada operação na latência total de indexação de um novo campo de texto. A alteração do esquema da base de dados para adicionar um novo campo adiciona uma latência muito residual ao processo e que não varia com a quantidade de informação que esta armazena. Já a operação de criação de um novo índice acrescenta uma latência proporcional com a quantidade de registos na base de dados, apesar de ser sempre muito inferior à operação final deste processo. A única operação comum a Solr, a inserção dos dados, é aquela que mais contribui para a latência final do teste.

A nível de espaço, o impacto da indexação do novo atributo por cada 1000 registos está representado na Figura 5.29. Aqui temos o tradicional trade-off. Oracle é aproximadamente 50% mais eficiente na indexação do que Solr a nível de espaço, no entanto como se pode constatar na Figura 5.27, Solr é claramente mais eficiente no que diz respeito às latências na indexação. Por outro lado, Solr gradualmente melhora a eficiência de indexação, embora no fim do teste o ritmo de melhoria seja baixo e já Oracle evidencia um comportamento com um aspecto mais constante, junto dos 50 KBytes por cada 1000 registos.

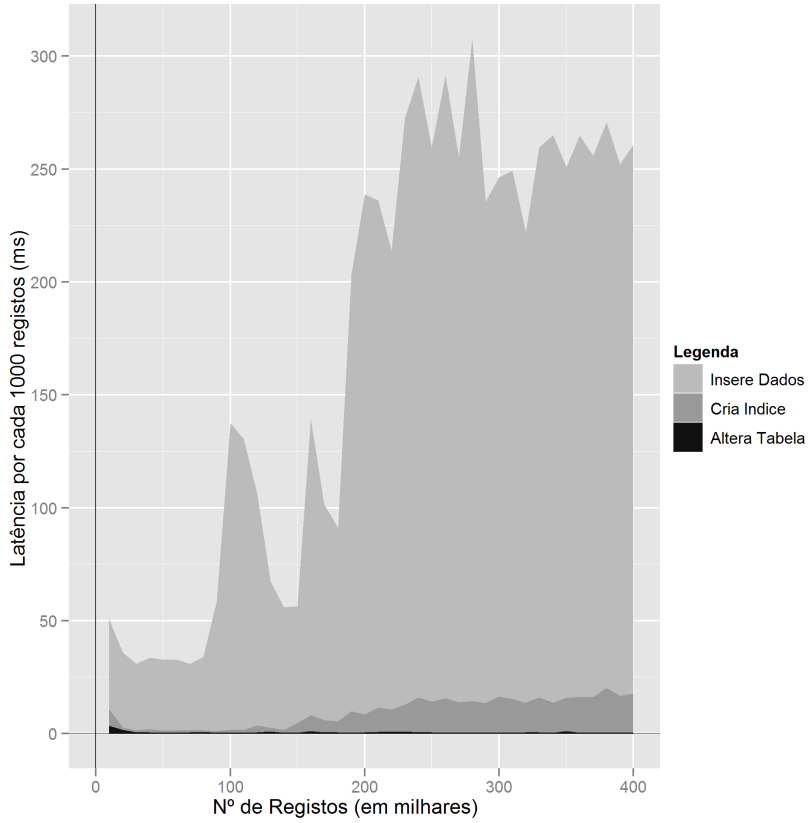



Figura 5.28: Detalhe das latências de criação de um índice em Oracle, discriminando as várias operações intermédias.

## 5.4 Conclusões

O *benchmark* realizado permitiu tirar conclusões claras. Na sua configuração de base, Solr tem um desempenho muito bom, mesmo com o aumento da exigência dos testes. A variação de factores como o número de atributos a indexar e a quantidade de documentos a inserir e indexar têm um impacto muito reduzido nas latências de resposta apresentadas por Solr. Esta plataforma mantém ainda o mesmo nível de desempenho nas inserções e pesquisas quando é utilizado um índice com um filtro divisor por espaços, que permite a pesquisa de palavras. Deste modo, ficou comprovada a capacidade desta plataforma para responder a cenários muito exigentes de indexação. Já Oracle, um sistema mais abrangente e poderoso, tem uma maior exigência a nível de recursos para manter um comportamento razoável à medida que os testes se tornam mais complexos. Oracle tem, nos cenários menos exigentes, uma performance superior à de Solr. No entanto, com o aumento da informação a indexar, as latências registadas em Oracle aumentam descontroladamente. Para melhorar o seu rendimento, era necessário refinar a sua poderosa configuração e adaptá-la ao ambiente onde é instalado, o que não está previsto neste trabalho

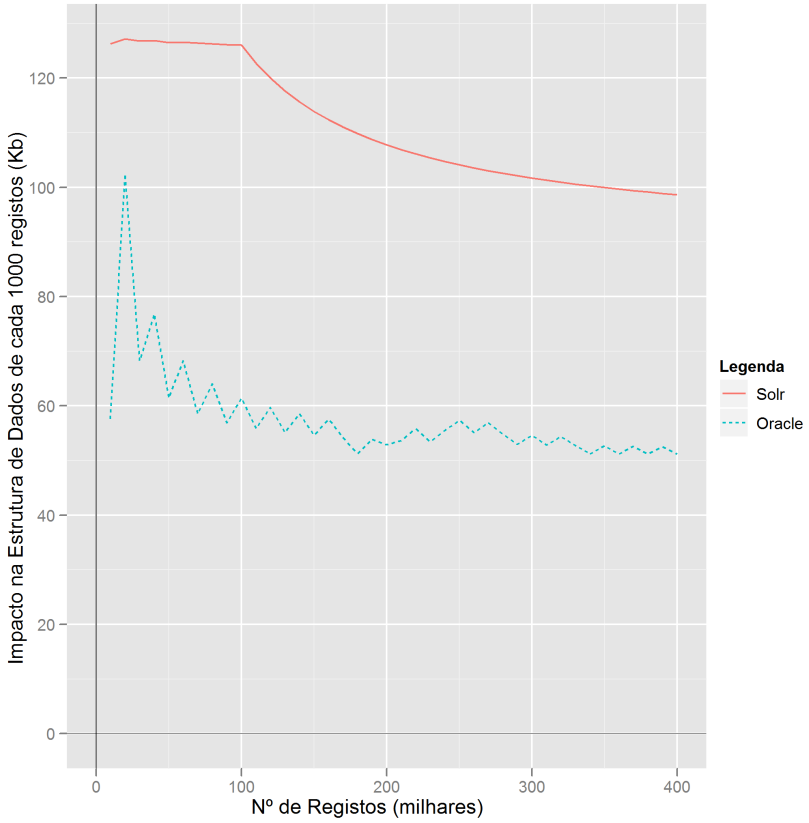



Figura 5.29: Impacto na dimensão da estrutura por cada 1000 registos inseridos.

para nenhuma das plataformas avaliadas.

Para definir um sistema que permita a pesquisa de palavras-chave eficiente em informação estruturada baseado numa única tecnologia exigiria uma adaptação muito intensiva ao domínio do problema. Esta metodologia é inviável para a maioria das organização, uma vez que ao adaptar o software a um domínio se perde a capacidade de reaproveitar software.

Solr é uma plataforma orientada para a indexação e não para o armazenamento de grandes quantidades de informação. Inclusivamente, foi criado para ser integrado com uma base de dados estruturada. Oracle é um sistema com um foco em áreas como o armazenamento, interrogação e processamento. Assim, uma arquitectura composta pela coexistência de um sistema relacional e de um sistema de pesquisa que siga num modelo não relacional, permite aproveitar as capacidades de armazenamento do primeiro e a orientação para a indexação do segundo. No caso específico dos sistemas Oracle e Solr, poderia pensar-se numa arquitectura onde Oracle guardava toda a informação inserida, mas apenas indexava um campo identificador de cada registo. Solr guardava e indexava toda a informação que podia ser pesquisada. Cada operação de inserção, actualização ou remoção teria de ser feita em Oracle e replicada em Solr. Já as consultas seriam efectua-

das em Solr, que por sua vez, retornava apenas os identificadores dos registos que fazem parte dos resultados. Estes identificadores eram úteis para em Oracle eficientemente obter todos os dados dos registos que iriam ser retornados ao utilizador.

# Capítulo 6

# Conclusões e Trabalho Futuro

## 6.1 Conclusões

A pesquisa por palavras-chave em base de dados estruturadas foi motivada pelo sucesso dos sistemas de pesquisa na *web*. A interface simples e amigável associada a estes sistemas destaca-se a nível da usabilidade, uma vez que proporciona que os termos da pesquisa sejam inseridos de uma forma muito natural.

Para definir um sistema que permita a pesquisa por palavras-chave em base de dados estruturadas, foram estudadas várias plataformas que se baseiam em paradigmas diferentes. Por um lado, temos os SGBD relacionais conhecidos pela pesquisa estruturada, e por outro, temos um conjunto vasto de plataformas vocacionadas para a pesquisa por texto livre, que pretende dar suporte à informação estruturada. Assim, nesta dissertação foram avaliadas duas plataformas, uma solução relacional e outra não relacional.

O método utilizado para averiguar as capacidades das plataformas consistiu na execução de uma bateria de testes que cobre os casos de uso mais relevantes na perspectiva da empresa PT Inovação. O sistema relacional Oracle apresentou-se claramente mais orientado para o armazenamento da informação tal como é reconhecido, e com uma exigência muito maior ao nível de recursos quando comparado com Solr, para indexar grandes quantidades de informação. A nível do tempo de resposta na pesquisa por palavras-chave os resultados obtidos estiveram bastante aquém da plataforma Solr. Para obter um desempenho aceitável nos testes mais exigentes Oracle reforçou a necessidade de refinamento na sua configuração, ao passo que Solr, um sistema baseado num modelo orientado ao documento, apresentou em todos os testes efectuados uma performance muito satisfatória e por norma, bastante superior à do Oracle. A orientação de Solr para a indexação é notória, uma vez que apresentou um desempenho praticamente inalterado com a variação da quantidade e complexidade da informação a indexar.

Por fim, para se obter resultados muito bons em áreas tão complexas como o armazenamento e a indexação utilizando uma única plataforma, é necessário que a sua configura-

ção seja muito adaptada ao ambiente onde a plataforma está inserida. É muito dispendioso por um lado, obter de um sistema relacional um bom rendimento na fase de indexação e por outro, obter de um sistema baseado num paradigma diferente bons resultados na área do armazenamento. Deste modo, uma arquitectura composta por um sistema relacional e por um sistema de pesquisa baseado num modelo não relacional, poderá permitir aproveitar as capacidades de armazenamento de um sistema relacional como o Oracle e a vocação para a indexação de um sistema baseado num modelo diferente, como o Apache Solr.

## 6.2 Trabalho Futuro

O trabalho realizado estuda exaustivamente as possibilidades de adoptar soluções baseadas ou não em modelos relacionais para dar resposta à pesquisa por palavras-chave em informação estruturada. Apesar de tudo, os resultados experimentais poderiam ter uma maior profundidade ao apresentar indicadores como a quantidade de memória principal e de CPU utilizado. Um outro indicador muito interessante é a eficácia da pesquisa, para verificar se de facto os primeiros resultados são os mais relevantes.

Por outro lado, os testes efectuados não contemplam optimizações na configuração nas plataformas. Para efeitos de testes, convém comparar as plataformas nas mesmas condições, o que não cria espaço a muitas optimizações. No entanto, para tornar o trabalho mais completo poderia no futuro efectuar-se testes numa situação onde as plataformas fossem optimizadas para obter o melhor rendimento possível. Deste modo, as plataformas seriam testadas num cenário próximo dos ambientes reais de produção, mas por outro lado, era sempre discutível a justiça do teste, ou seja, se de facto, as plataformas estavam com o mesmo nível de refinamento.

Outra perspectiva de trabalho futuro, consiste no alargamento da gama de sistemas avaliados. Os sistemas estudados Oracle e Solr não representam totalmente os sistemas relacionais e os baseados em modelos não relacionais, respectivamente. Assim, e para tornar o trabalho mais abrangente, a bateria de testes utilizada neste trabalho poderia futuramente ser utilizada para avaliar outros sistemas, nomeadamente o sistema Sphinx Search.

# Referências


[ACD02] Sanjay Agrawal, Surajit Chaudhuri e Gautam Das. Dbxplorer: A system for keyword-based search over relational databases. *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[ACDG03] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das e Aristides Gionis. Automated ranking of database query results. *Proceedings of the 2003 CIDR Conference*, 2003.

[Aks10] Andrew Aksyonoff. Meet the sphinx. *DORS/CLUC*, 2010.

[BHNC02] Gaurav Bhalotia, Arvind Hulgeriy, Charuta Nakhez e Soumen Chakrabarti. Keyword searching and browsing in databases using banks. In *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[Bia08] Andrzej Bialecki. Luke - lucene index toolbox, Novembro 2008. Disponível em `http://www.getopt.org/luke/`, acedido última vez em 17 de Junho de 2011.

[BS09] Oleg Bartunov e Teodor Sigaev. Some recent advances in full-text search. Technical report, Sternberg Astronomical Institute, Moscow State University, Maio 2009.

[Bur] Don Burleson. Oracle like clause - remove full table scans with text indexes. Disponível em `http://www.dba-oracle.com/oracle_tips_like_sql_index.htm`, acedido última vez em 17 de Junho de 2011.

[Bur11] Don Burleson. Oracle consulting - inserts become slow as table grows, Abril 2011. Disponível em `http://www.dba-oracle.com/t_inserts_become_slow_as_table_grows.htm`, acedido última vez em 17 de Junho de 2011.

[Car11] Nicholas Carlson. Facebook has more than 600 million users, Janeiro 2011. Disponível em `http://www.businessinsider.com/facebook-has-more-than-600-million-users-goldman-tells-clients-2011-1`, acedido a última vez em 17 de Junho de 2011.

[CI98] Chee-Yong Chan e Yannis E. Ioannidis. Bitmap index design and evaluation. *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, Junho 1998.

[Cor] Oracle Corporation. The world's most popular open source database. Disponível em `http://www.mysql.com`, acedido a última vez em 17 de Junho de 2011.

[Cor03] Oracle Corporation. *Oracle Database Performance Tuning Guide and Reference*. Dezembro 2003.

[Cor05] Microsoft Corporation. Usability in software design, Janeiro 2005. Disponível em `http://msdn.microsoft.com/en-us/library/ms997577.aspx`, acedido a última vez em 17 de Junho de 2011.

[Cor09a] Oracle Corporation. Oracle database reference - dba_segments, 2009. Disponível em `http://download.oracle.com/docs/cd/B19306_01/server.102/b14237/statviews_4097.htm`, acedido última vez em 17 de Junho de 2011.

[Cor09b] Oracle Corporation. *Oracle Text Reference 11g*. Julho 2009.

[Cor10] IBM Corporation. *DB2 Version 9.5 Text Search Guide*. Dezembro 2010.

[Cut05] Doug Cutting. Open source search, Dezembro 2005. Information Retrieval Seminar - IBM Research Lab.

[CWLL09] Yi Chen, Wei Wang, Ziyang Liu e Xuemin Lin. Keyword search on structured and semi-structured data. In *Proceedings of the 35th SIGMOD international conference on Management of data*, pages 1005–1010, 2009.

[Fer08] Fabrício Ferrari. A importância da usabilidade, Outubro 2008. Disponível em `http://qualidadebr.wordpress.com/2008/10/01/a-importancia-da-usabilidade`, acedido a última vez em 17 de Junho de 2011.

[Fou06] The Apache Software Foundation. Welcome to solr, Janeiro 2006. Disponível em `http://lucene.apache.org/solr/`, acedido a última vez em 17 de Junho de 2011.

[Fou09] The Apache Software Foundation. Welcome to apache lucene!, Janeiro 2009. Disponível em `http://lucene.apache.org/`, acedido a última vez em 17 de Junho de 2011.

[Fou11] The Apache Software Foundation. Solrj, Abril 2011. Disponível em `http://wiki.apache.org/solr/Solrj`, acedido última vez em 17 de Junho de 2011.

[Fun11] The Apache Software Fundation. Apache tomcat, 2011. Disponível em `http://tomcat.apache.org/`, acedido última vez em 17 de Junho de 2011.

[Gar07] Jessica Gardner. Remote usability testing of the unece website. *Meeting on the Management of Statistical Information Systems*, 2007.

[Goo] Google. Descrição geral da tecnologia. Disponível em `http://www.google.com/corporate/tech.html`, acedido a última vez em 17 de Junho de 2011.

[Hay04] Martin Haye. Cross-instance search system: Search engine comparison. Janeiro 2004.

[HFBYL92] D. Harman, E. Fox, R.A. Baeza-Yates e W. Lee. Inverted files. *Information Retrieval: Data Structures and Algorithms*, pages 28–43, 1992.

[HGP03] Vagelis Hristidis, Luis Gravano e Yannis Papakonstantinou. Efficient ir-style keyword search over relational databases. In *Proceedings of the 29th International VLDB Conference*, pages 850–861, 2003.

[HP02] Vagelis Hristidis e Yannis Papakonstantinou. Discover: Keyword search in relational databases. *Proceedings of the 28th VLDB Conference*, 2002.

[Kar09] Bill Karwin. Practical full-text search in mysql, Dezembro 2009. MySQL University.

[LC86] Tobin J. Lehman e Michael J. Carey. A study of index structures for main memory database management systems. *Proceedings of the 12th International VLDB Conference*, 11(1):294–303, 1986.

[LNT00a] Hongjun Lu, Yuet Yeung Ng e Zengping Tian. Random words, quantum statistics, central limits, random matrices. *Methods Appl. Anal.*, 9(1):101–109, 2000.

[LNT00b] Hongjun Lu, Yuet Yeung Ng e Zengping Tian. T-tree or b-tree: Main memory database index structure revisited. *Proceedings of the Australasian Database Conference*, 11(1):309–312, 2000.

[Lop08] Fernando Azpeitia Lopez. Sql server 2008 full-text search: Internals and enhancements. Technical report, Microsoft Corporation, Julho 2008.

[LWL08] Yi Luo, Wei Wang e Xuemin Lin. Spark: A keyword search engine on relational databases. Technical report, School of Computer Science and Engineering University of New South Wales Australia, 2008.

[LYMC06] Fang Liu, Clement Yu, Weiyi Meng e Abdur Chowdhury. Effective keyword search in relational databases. *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 2006.

[MBY07] Christian Middleton e Ricardo Baeza-Yates. A comparison of open source search engines. Technical report, Universitat Pompeu Fabra, Outubro 2007.

[Not08] Mark Nottingham. Http benchmark rules, Julho 2008. Disponível em `http://www.mnot.net/blog/2011/05/18/http_benchmark_rules`, acedido última vez em 17 de Junho de 2011.

[Pat08] Lorraine Paterson. New facebook: Usability research on a grand scale, Julho 2008. Disponível em `http://user-vision.blogspot.com/2008/07/new-facebook-usability-research-on.html`, acedido a última vez em 17 de Junho de 2011.

[Sch] Roger Schrag. Tuning oracle's buffer cache. Disponível em `http://www.dbspecialists.com/files/presentations/buffercache.html`, acedido última vez em 17 de Junho de 2011.

[See06] Yonik Seeley. Apache solr. *Apachecon Europe 2006*, 2006.

[Ser09] Sphinx Open Source Search Server. Sphinx technologies inc, 2009. Disponível em `http://sphinxsearch.com/`, acedido a última vez em 17 de Junho de 2011.

[Sin09] Vik Singh. A comparison of open source search engines, Julho 2009. Disponível em `http://zooie.wordpress.com/2009/07/06/a-comparison-of-open-source-search-engines-and-indexing-twitter`, acedido última vez em 17 de Junho de 2011.

[SN05] J. E. Smith e R. Nair. The architecture of virtual machines. *Computer*, 38:32–38, Maio 2005.

[Sol11] Solr. Solr performance factors, Junho 2011. Disponível em `http://wiki.apache.org/solr/SolrPerformanceFactors`, acedido última vez em 17 de Junho de 2011.

[SOT+00] T. Suganuma, T. Ogasawara, M. Takeuchi, T. Yasue, M. Kawahito, K. Ishizaki, H. Komatsu e T. Nakatani. Overview of the ibm java just-in-time compiler. *IBM Syst. J.*, 39:175–193, 2000.

[Str07] Martin Streicher. Build a custom search engine with php. *Linux Magazine*, Julho 2007.

[SW05] Qi Su e Jennifer Widom. Indexing relational database content offline for efficient keyword-based search. In *Proceedings of the 9th International Database Engineering & Application Symposium*, pages 297–306, 2005.

[WLK03] Richard Wheeldon, Mark Levene e Kevin Keenoy. Search and navigation in relational databases. Technical report, Birkbeck University of London, 2003.

[ZM06] Justin Zobel e Alistair Moffat. *Inverted Files for Text Search Engines*. Information Retrieval, Julho 2006.

[ZW07] Jiang Zhan e Shan Wang. Itreks: Keyword search over relational database by indexing tuple relationship. In *Proceedings of the 12th international conference on Database systems for advanced applications*, pages 67–78. Springer-Verlag, 2007.

# Anexo A

# Gráficos

## A.1 Caso de uso "inserir"

### A.1.1 Teste dos Registos

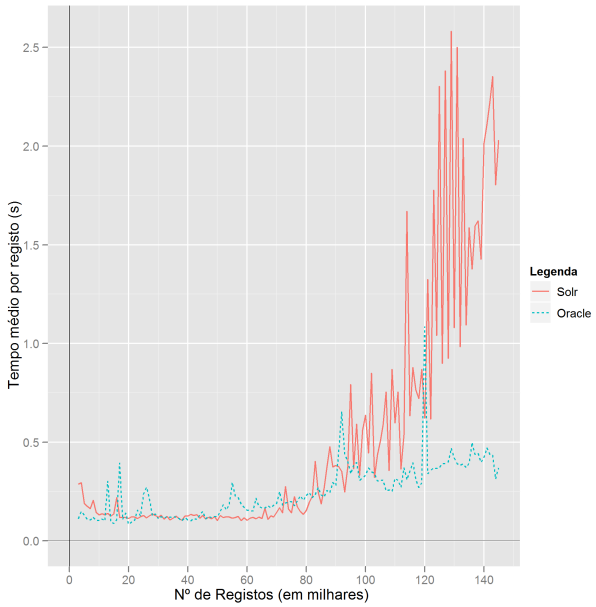



Figura A.1: Latência média de inserção de cada registo à medida que aumenta o número de registos em cada *batch*, referente ao Cenário II.

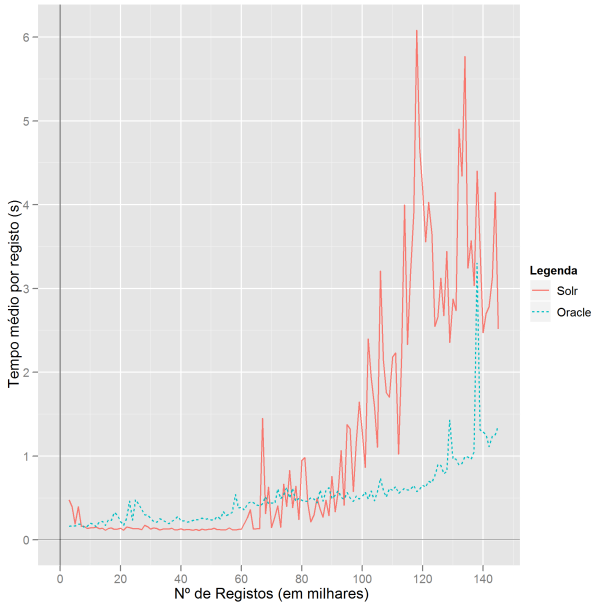



Figura A.2: Latência média de inserção de cada registo à medida que aumenta o número de registos em cada *batch*, referente ao Cenário III.

### A.1.2 Teste dos Campos de Texto

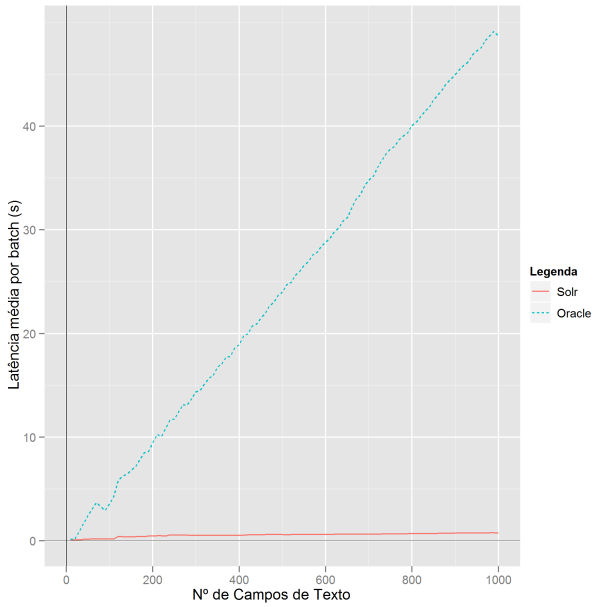



Figura A.3: Latência média de inserção de cada *batch* à medida que aumenta o número de campos de texto da base de dados, referente ao Cenário II.

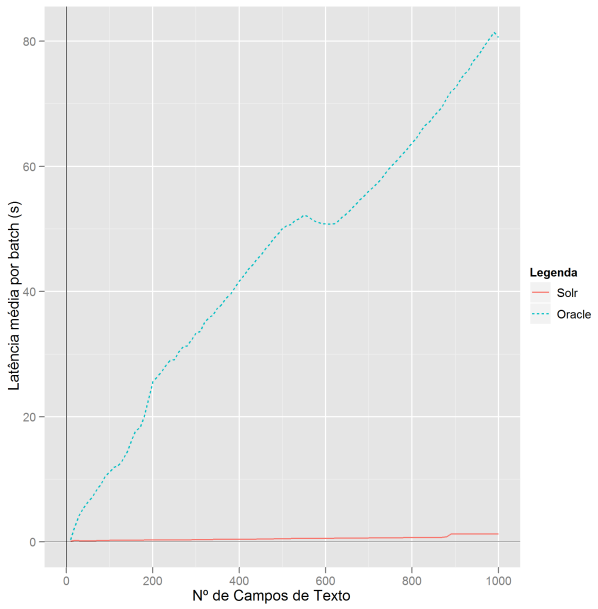



Figura A.4: Latência média de inserção de cada *batch* à medida que aumenta o número de campos de texto da base de dados, referente ao Cenário III.

## A.2 Caso de uso "pesquisar"

### A.2.1 Teste dos Registos

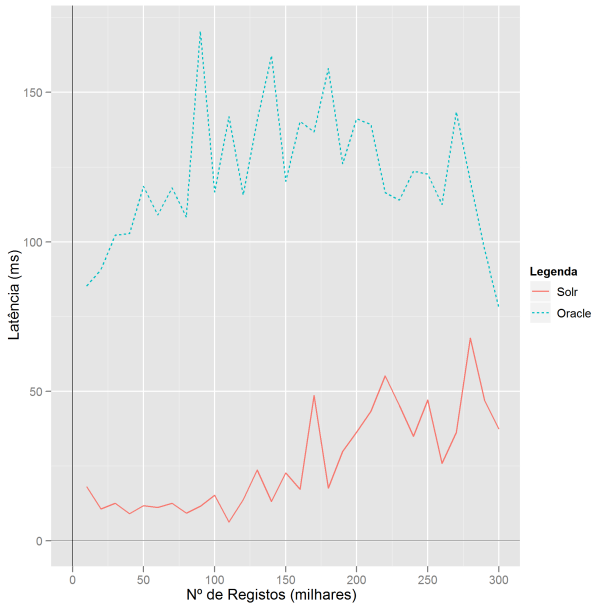



Figura A.5: Latência média de cada consulta à medida que as plataformas acumulam informação, referente ao Cenário III.

### A.2.2 Teste dos Campos de Texto

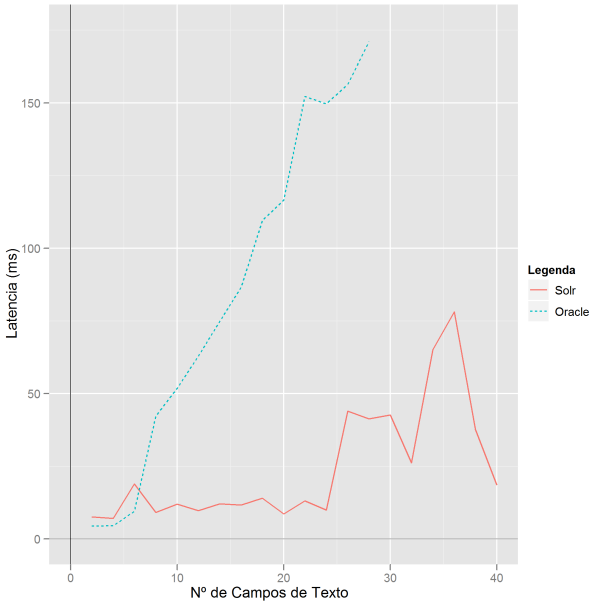



Figura A.6: Latência média de cada consulta à medida que aumenta o número de campos de texto na base de dados, referente ao Cenário III.

### A.2.3 Teste dos Resultados

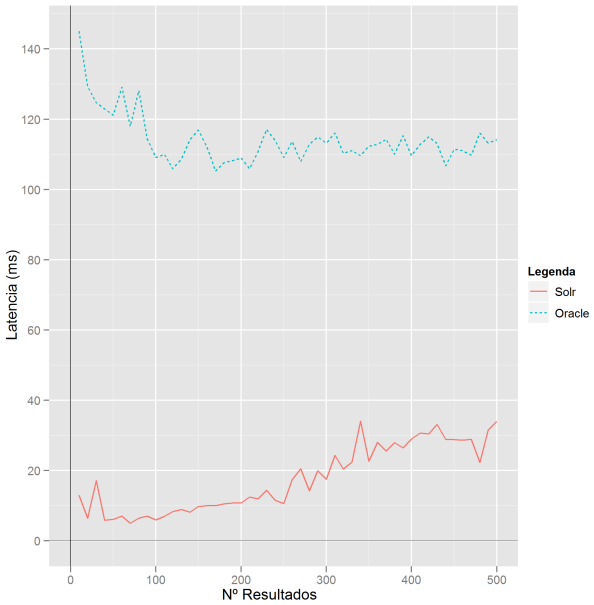



Figura A.7: Latência média de cada consulta à medida que o número de resultados que cada consulta devolve no máximo aumenta, referente ao Cenário III.

### A.2.4 Teste da Carga

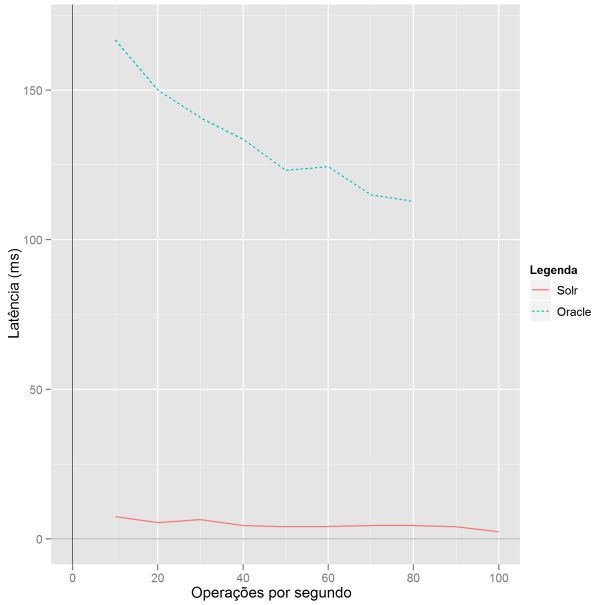



Figura A.8: Latência média de cada consulta à medida que a carga aumenta nas plataformas, referente ao Cenário III.