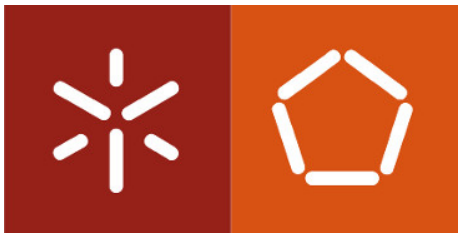
Laboratórios de Informática III

Trabalho Prático

Linguagem Java

Grupo nr. 4

a84485	Tiago Magalhães
a85517	Duarte Oliveira
a89512	Manuel Novais



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1 Introdução

O projeto da disciplina de Laboratórios de Informática III visava a consolidação a nível teórico-prático de conhecimentos previamente lecionados em unidades curriculares deste curso. Para além do desenvolvimento de capacidades a nível individual, também se procurava introduzir novos princípios e bons hábitos na área de Engenharia de Software. Por conseguinte, esperava-se um aprofundamento dos conhecimentos associados à manipulação de grandes volumes de dados ao mesmo tempo que, e intrinsecamente, se inseria uma maior complexidade algorítmica para o tratamento destes.
Do início ao fim a evolução deste projeto acompanhou as ideias e os conhecimentos passados nas aulas práticas pelos docentes, tendo em conta que os membros deste grupo possuíam muito pouco conhecimento em relação a esta área da engenharia informática e foi graças à abordagem pedagógica que conseguiu-se desenvolver certas partes do mesmo.
Tal como na fase anterior, quando foi dado o acesso aos ficheiros de dados (Produtos, Clientes e Vendas), conseguimos criar uma imagem mais concreta da imensidão e da complexidade característica dos códigos de produtos, de clientes e de vendas. Tornou-se assim claro o porquê da necessidade primordial de uma codificação que favorecesse a performance mas também a sua confiabilidade. Ou seja, com um grande volume de dados há a clara necessidade de proceder à sua verificação de uma forma, no mínimo, razoável, evitando assim que o seu tempo de leitura seja pouco aprazível para o ser humano e que não consuma recursos desnecessários da máquina que os trata.

Em retrospetiva concluímos assim o quão necessária foi a adoção de diferentes abordagens que permitissem tirar o melhor partido possível das capacidades de execução e de iteração que a linguagem JAVA nos proporciona.

1.1 Problema e Objetivo

Como referido anteriormente, o problema levantado para este projeto era a gestão de um sistema de vendas, possivelmente uma cadeia empresarial com um vasto leque de recursos a serem vendidos em pelo menos três filiais e com um extenso número de clientes, todos devidamente identificados. Nesta ótica, havia também um conjunto de vendas que representava a compra de um produto por um determinado cliente. Essa compra tinha em si identificada a quantidade adquirida, se a compra tinha sido feita em promoção ou não e em qual filial é que tinha ficado registada a compra, para além do cliente e do produto devidamente assinalados. É de notar que havia a perceção de que dentro dos dados haveria a existência de entradas inválidas, ou seja, códigos de produtos ou de clientes inválidos. Para além disso, tanto os clientes poderiam estar registados sem terem feito qualquer tipo de compra como também os produtos podiam estar identificados sem fazerem parte de qualquer tipo de vendas. Era essencial saber percorrer estes dados e selecionar os que realmente convinham para .
Após esta verificação, era necessário elaborar um conjunto de características que este sistema devesse conter, de modo a que a manipulação e o tratamento de dados pudesse invocar valores em concreto, em correlacionados com os dados disponibilizados. Com isto remete-se para um dos principais e mais trabalhosos objetivos do trabalho: a construção e desenvolvimento de *queries* apresentadas pela equipa docente. Nesta fase, tanto denominadas como "*queries* estatísticas"e "*queries* iterativas"tendo sempre presente que havia a necessidade de manter o histórico de vendas organizado por filiais.

Finalmente esperava-se uma arquitetura conveniente, que de modo evolutivo e individual fosse continuamente testada, para se proceder a uma mais completa implementação e finalmente uma total acoplação, dando origem a um programa devidamente estruturado - arquitetura *Model View Controller*.
Estes e outros aspetos irão ser tratados neste manuscrito de forma mais detalhada, para uma melhor perceção do leitor.

1.2 Estrutura do Relatório

O relatório apresenta-se de forma detalhada. Cada tópico remete para uma parte em específico que foi abordada e tratada no nosso projeto. Primamos pelo seguimento delineado do que foi sugerido pelos docentes desta unidade curricular de modo a varrer o maior número de tópicos possível. Esses tópicos são devidamente descritos e identificados ao longo do comento. Optamos pelo uso de imagens tanto da esquematização de estruturas de dados como de blocos de código para corroborar as nossas afirmações e para mais facilmente fazermos compreender a nossa perspetiva e o nosso planeamento para este projeto. Na conclusão fazemos uma retrospetiva deste projeto, apresentando problemas, dificuldades, ambições e também possíveis melhorias.

2 Concepção/Desenho da resolução

2.1 Estruturas de Dados

Aqui será abordado o esqueleto por nós usado para a implementação de informação neste projeto.

Estruturas onde são carregados os dados:

- Catálogo de Produtos
- Catálogo de Clientes
- Faturação
- Filial

2.1.1 Catálogos

Nos catálogos de Produtos e Clientes, optámos por guardar tanto os códigos dos produtos como o dos clientes num *TreeSet*, uma vez que existia a necessidade de listagem de códigos em algumas das *queries* terem de ser feitas por ordem alfabética e não existir necessidade de guardar informação repetida. Além desta estrutura permitir uma inserção ordenada alfabeticamente e mesmo havendo a necessidade de verificação nas vendas para ver se um produto ou cliente pertence ao seu respetivo catálogo, apesar do TreeSet não ter um tempo de acesso direto, a sua complexidade continua a ser razoável $O(\log n)$ tendo em conta os propósitos deste projeto.

Estruturas dos Catálogos:

```
private Set<ICliente> clientes;

private Set<IProduto> produtos;
```

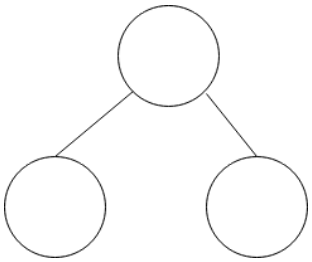


Figura 1: Representação gráfica dos catálogos

2.1.2 Faturação

Dentro da estrutura faturação optámos pelo uso de uma List que representa os meses, cada mês tem um HashMap em que a key é o produto e o value é uma InfoFat, sendo está constituida por um HashMap que corresponde às filiais e uma List SimpleEntry ¡Integer, Float¿, que representa as vendas desses produtos guardando um par preço/quantidade da venda. Optámos por esta estrutura com HashMaps para permitir ter um acesso direto, dado que ao ler o ficheiro vendas teríamos de inserir informação nova. A estruturação encontra-se dividida por filiais e meses uma vez que existiam muitas queries a pedir a faturação numa determinada filial e mês.

Estrutura da Faturação:

```
/* Estrutura da Faturação */

/* Lista Meses -> Map( Key : Produto Value : InfoFat ) */
private List<Map<IProduto, InfoFat>> faturacao;

/* Estrutura da InfoFat */

/* Map Filial -> Key : NmrFilial Value : unidades,preco  */
private Map<Integer, List<SimpleEntry<Integer, Float>>> fat;
```

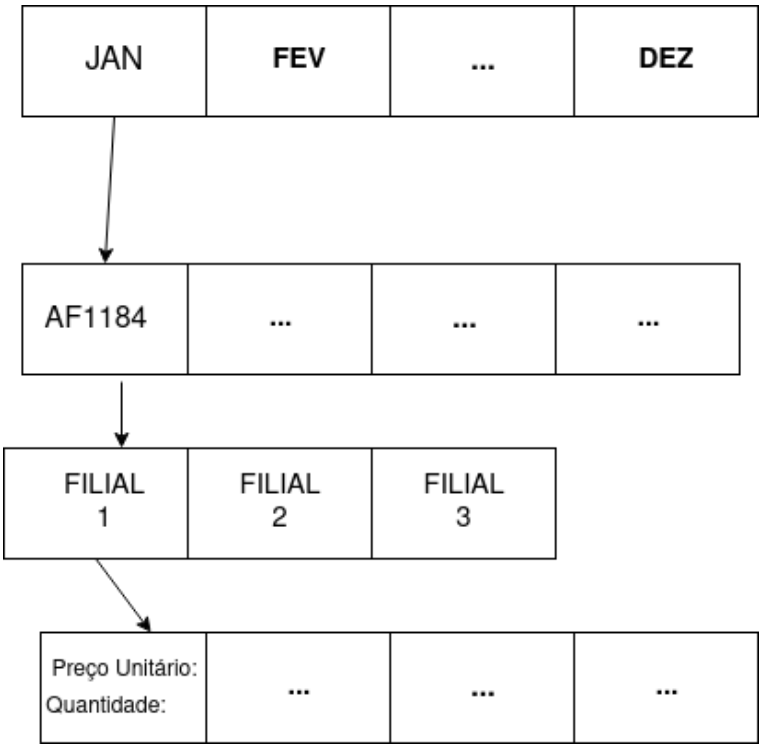


Figura 2: Representação gráfica faturação

2.1.3 Filial

Na filial, tal como na faturação, devido à necessidade de atualização de informação, optámos pelo uso de *HashMaps's*, nas filial este HashMap tem como key o cliente e como value uma lista de compras do cliente, que guarda o produto, preço, quantidade, tipo de compra e mês da compras.

Estrutura da Filial:

```
/* Estrutura de uma Filial */
// Map de clientes com uma lista das compras do cliente
private Map<ICliente, List<InfoFilial>> filial;

/* Estrutura InfoFilial */
/* Varáveis de instância */
private IProduto produto;
private float preco;
private int quant;
private char tipoCompra;
private int mes;
```

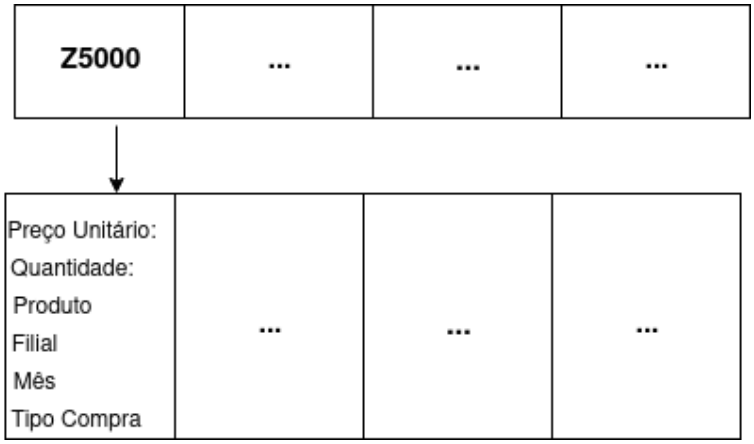


Figura 3: Representação gráfica filial

2.2 Estruturas Auxiliares

Como estruturas auxiliares, usamos triplos e pares de auxílio à resolução das queries.

2.3 Modularidade

O projeto encontra-se distribuído por packages e também possui o uso de interfaces, visto que uma vez que trabalha com grande volume de dados, torna-se fundamental organizar o programa desta forma de forma a também poder existir a reutilização de código. As próprias classes intrínsecas ao Java já fornecem modularidade. A utilização de packages também facilita com que várias pessoas possam implementar diferentes partes do projeto, agilizando o nosso processo de manutenção, reconstrução e deteção de erros em código.O nosso projeto segue a arquitetura *Model-View-Controller* (MVC), contendo os seguintes packages presentes no projeto:

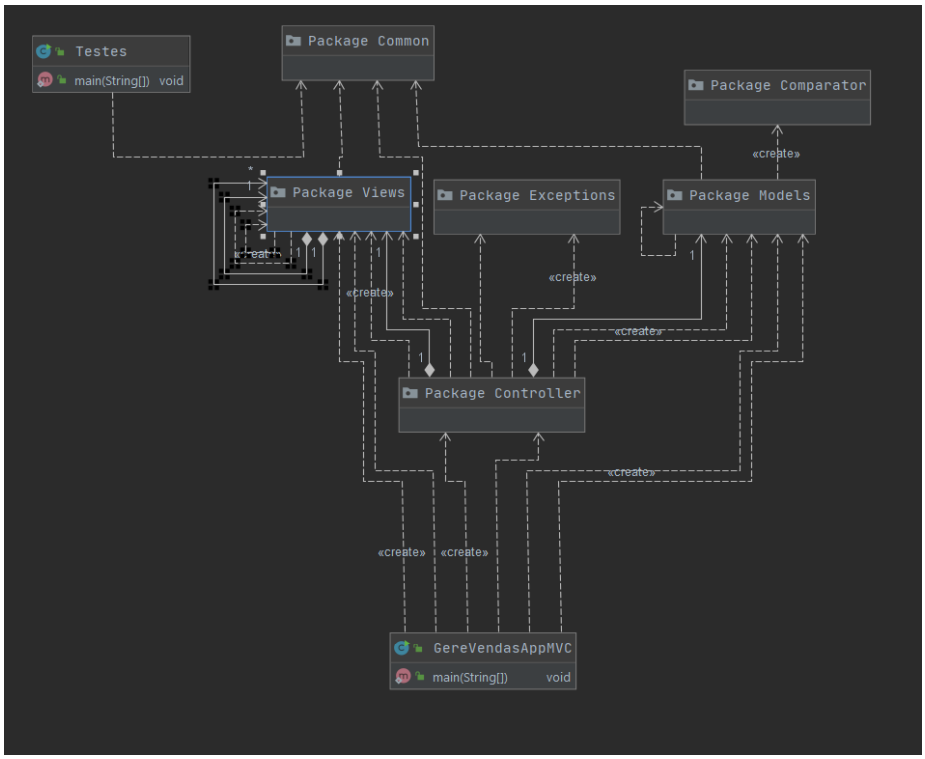


Figura 4: Packages

2.4 Encapsulamento

O uso da keyword *private* que torna os métodos/variáveis de instância acessíveis apenas na classe onde estão declaradas.

Figura 5: Método privado ao módulo

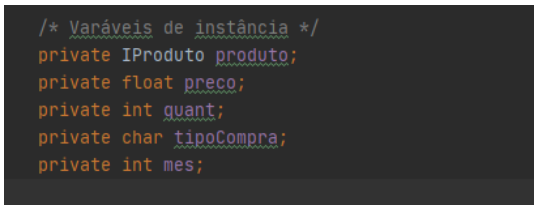


Figura 6: Variável de instância privada

O uso de *"clones"* para retornar uma cópia do objeto.

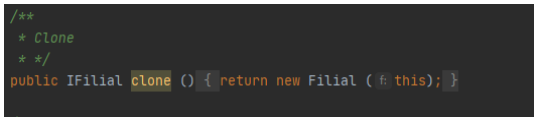


Figura 7: Clone

O encapsulamento também foi garantido,através de *"getters"*, assim as váriaveis de instância só são acedidos através destes métodos e não diretamente fora das suas classes.

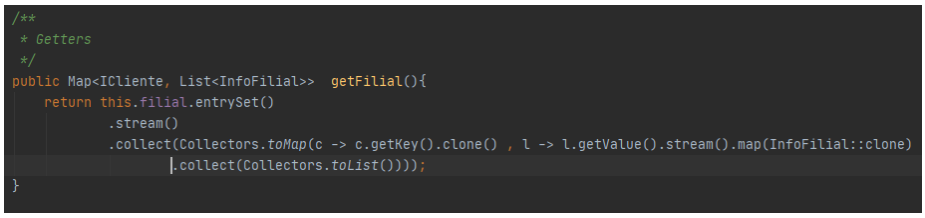


Figura 8: Getter

3 Medidas de performance/Testes de desempenho

3.1 Tempos de leitura com BufferedReader e files

Ficheiro	BufferedReader	Files
Vendas 1Milhão Sem parsing	0.1107246 s	0.1000037 s
Vendas 1Milhão Com parsing	0.4998611 s	0.38528446 s
Vendas 1Milhão Com parsing e validação	0.9690246 s	1.0841897 s

Ficheiro	BufferedReader	Files
Vendas 3Milhões Sem parsing	0.29290779 s	0.1547308 s
Vendas 3Milhões Com parsing	1.2901279 s	1.1007386 s
Vendas 3Milhões Com parsing e validação	2.8622239 s	3.1027664 s

Ficheiro	BufferedReader	Files
Vendas 5Milhões Sem parsing	0.347231 s	0.3028312 s
Vendas 5Milhões Com parsing	2.0206645 s	1.8273576 s
Vendas 5Milhões Com parsing e validação	5.0287575 s	5.2659641 s

3.2 Tempos Queries

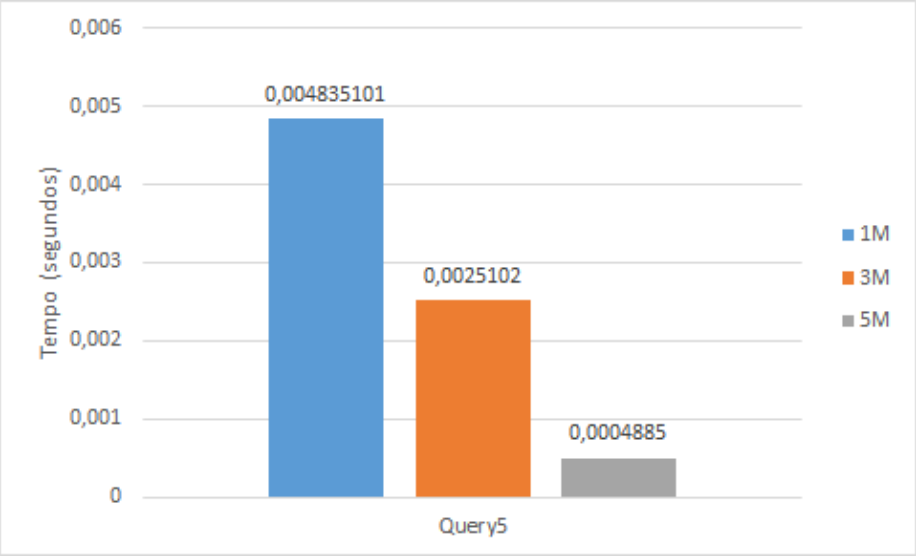


Figura 9: Query 5

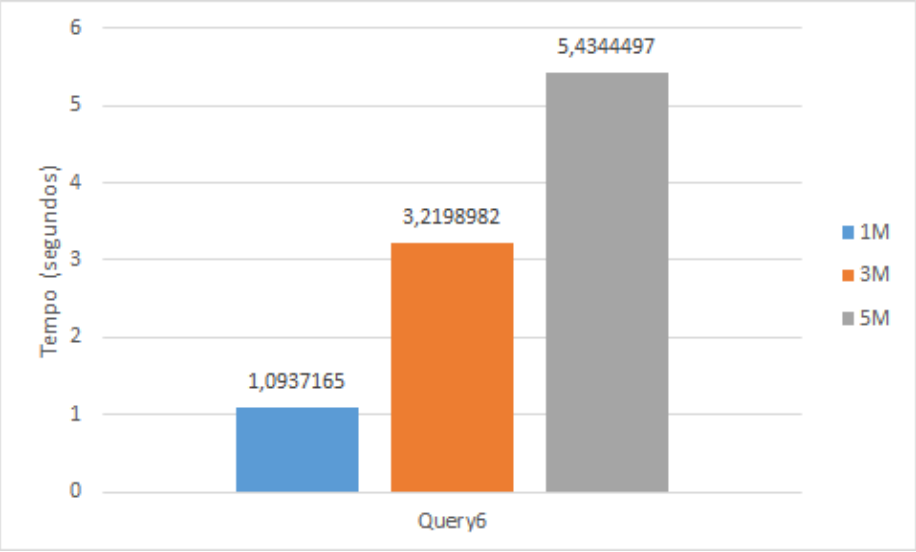


Figura 10: Query 6

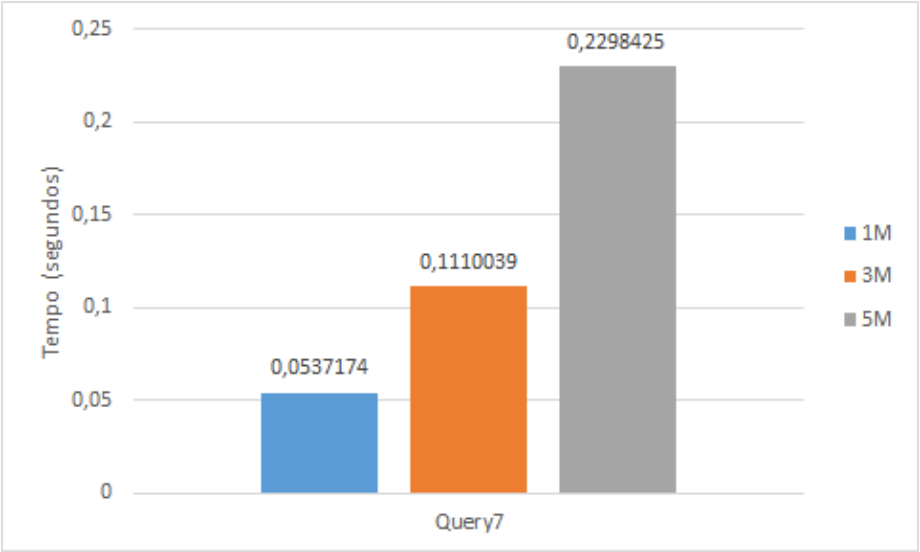


Figura 11: Query 7

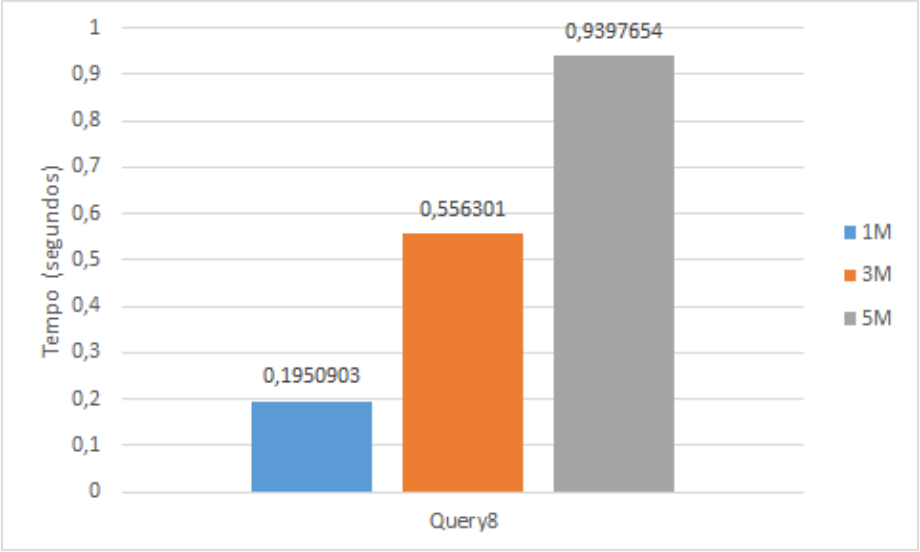


Figura 12: Query 8

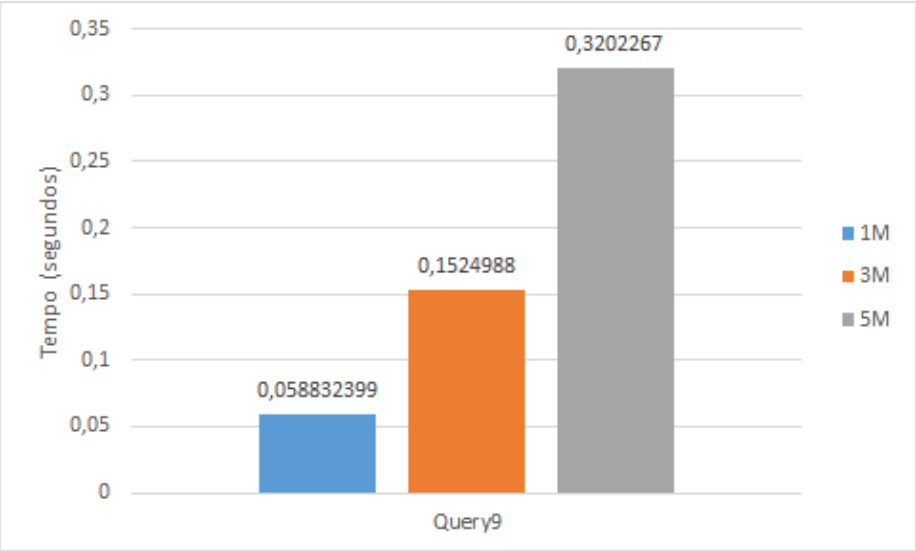


Figura 13: Query 9

Estes testes foram realizador num computador com as seguintes especificações:

4 Dificuldades

Em contraste com o projeto anterior, este trouxe-nos muito menos dificuldades técnicas, para além de haver alguma preferência pessoal pela linguagem Java, também esta trouxe-nos muitos menos problemas de compilação. Houve uma confusão inicial ao fazer *commits* por causa de alguns ficheiros residuais que o *IDE* de cada um gerava, mas após isso não se denotou qualquer problema que merecesse qualquer tipo de apreciação.

5 Conclusão

Com a realização deste trabalho em Java, notamos que em relação ao trabalho em C os tempos de leitura foram mais rápidos, isto deveu-se ao facto de usarmos estruturas menos complexas para guardar a informação, estas foram menos complexas uma vez que o Java possui nas API's de cada uma das estruturas vários métodos para as manipular das mais variáveis formas, o que nos permitia várias abordagens ao mesmo problemas,por conseguinte permitiu-nos guardar em estruturas menos complexas e depois fazer os cálculos necessários nas queries.Para um problemas de grande escala e com necessidade de testar várias estruturas o java devido à utilização de interfaces permite alterar estruturas sem necessidade alterar os métodos, o que faz com que exista uma grande reutilização de código.

A nível de melhoramento tentamos com que a query 6 se tornasse mais eficiente através do uso de paralelismo, no entanto não houve assim um relevante melhoramento, apesar de ter baixado um bocado o tempo.