Sistema de Gestão de Vendas

Trabalho realizado por:

João Figueiredo Martins Peixe dos Santos

Luís Filipe Cruz Sobral

Paulo Silva Sousa

A89520
João Santos

A89474
Luís Sobral

A89465
Paulo Sousa

Grupo 70
Projeto LI3
2019/2020
Universidade do Minho

Índice

1 Introdução

Nesta unidade curricular foi-nos proposta a implementação de um sistema de resposta a queries sobre um Sistema de Gestão de Vendas - **SGV**.

Inicialmente, o projeto consistiu no desafio de implementar este sistema na linguagem Java, já depois de o termos feito em C. Apesar da intenção de tornar rápida a execução das funcionalidades do programa a ser desenvolvido, alguns dos focos prioritários deste projeto foram a modularidade e o encapsulamento das estruturas de dados por nós utilizadas.

Ao longo do desenvolvimento deste projeto, consideramos que o maior desafio foi a escolha das estruturas onde seria organizada a informação, de forma a que o seu acesso fosse rápido e eficiente.

2 Escolha da Solução Adotada

Inicialmente, a estrutura que optamos por utilizar para a Filial e para a Faturação assemelhava-se à estrutura que ultilizamos no projeto em C (um HashMap de 26 posições de ArrayLists, onde cada posição de ArrayList tinha informação relativa a um Cliente/Produto). Com a necessidade de melhorar a performance do programa, começamos a pensar em utilizar estruturas que fossem mais eficientes em Java no âmbito deste projeto.

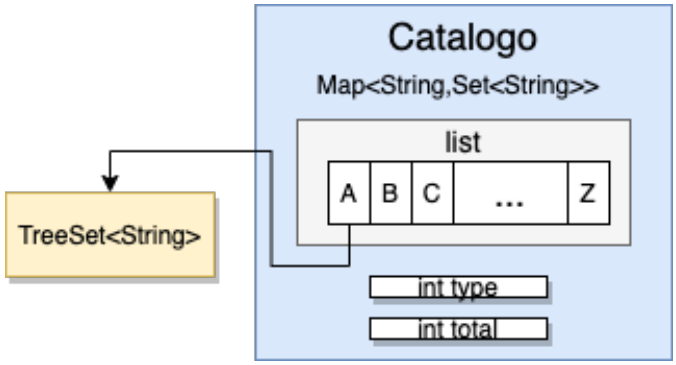
Em relação à classe Catalogo, usada quer para os Clientes quer para os Produtos, optamos pela utilização de um Map(HashMap) de 26 posições em que cada posição continha um Set(TreeSet) de Strings começadas pela letra associada à key do Map. Esta implementação permite-nos aceder a um Produto/Cliente em tempo logarítmico em função do número de elementos começados por uma letra, em vez do número total de Clientes/Produtos.

Em seguida, adaptamos as informações guardadas para cada Cliente/Produto de forma a poder resolver mais facilmente as queries.

Como não estavamos satisfeitos com o tempo de carregamento das estruturas alteramos a classe Filial e Faturacao para um Map(HashMap) onde a cada código de Cliente/Produto corresponde às informações relativas a este. Esta alteração permitiu-nos aceder à informação de um Cliente/Produto em tempo constante e diminuir o tempo de carregamento destas estruturas, uma vez que verificávamos se o Cliente/Produto já existia em tempo constante e, caso não existisse, adicionavamos um MapEntry. Na Filial, optamos por dividir os Clientes e os Produtos em dois Map(HashMap) de modo a poder aceder às informações de um Cliente/Produto mais facilmente.

3 Estruturas de Dados

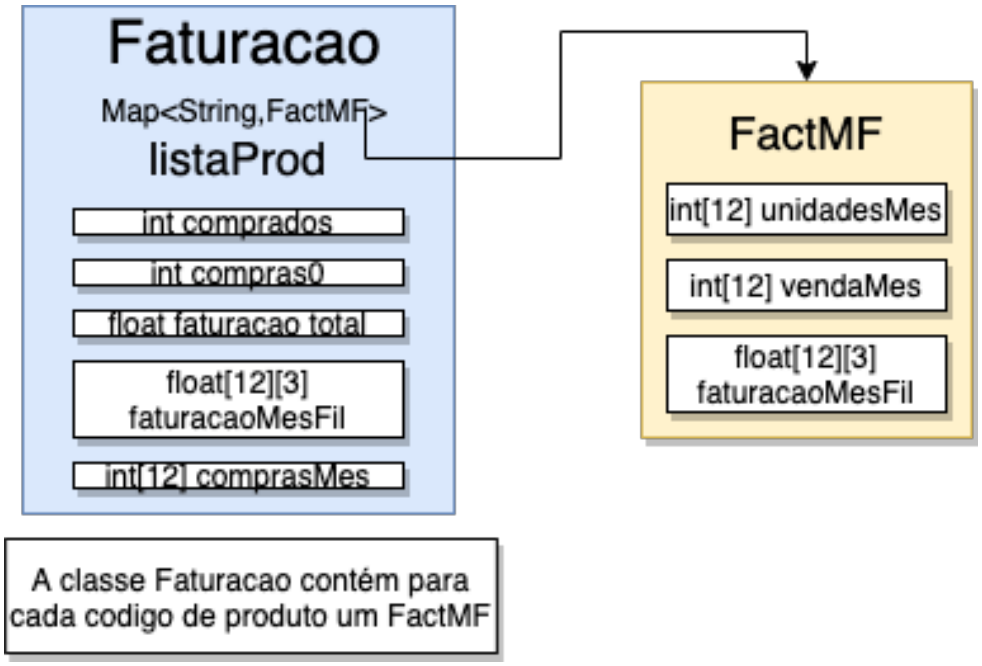
3.1 Catalogo



Esta é uma classe utilizada para guardar os códigos de clientes ou produtos dos dados fornecidos pelo utilizador.

A cada key corresponde uma letra do alfabeto.

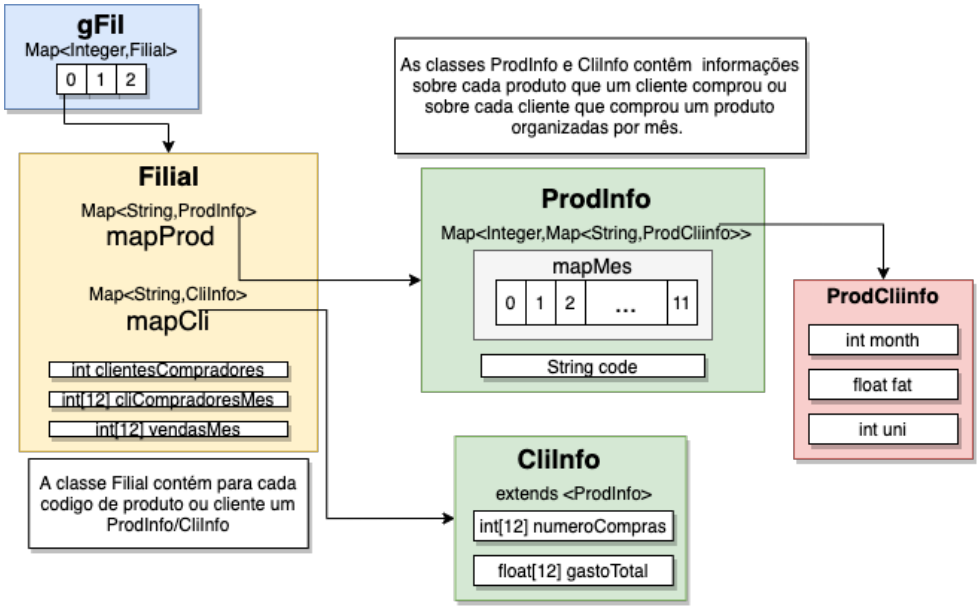
3.2 Faturacao



Esta classe é utilizada para armazenar as informações relativas à faturação de todas as vendas.

Nesta estrutura a cada key(correspondente a um código de produto) está associado a classe FactMF. Esta classe guarda as unidades e a vendas de cada mês e a faturação por mês/Filial. A classe Faturacao guarda um número de produtos comprados, o número de vendas de valor zero, a faturação total(global), a faturação total organizada por mês e Filial e o número de compras em cada mês.

3.3 GestaoFiliais



Esta é a classe utilizada para armazenar informação das vendas para cada filial.

Cada código de Produto que foi comprado numa Filial é representado por uma key no mapProd(HashMap) à qual corresponde as informações sobre as suas vendas(ProdInfo).

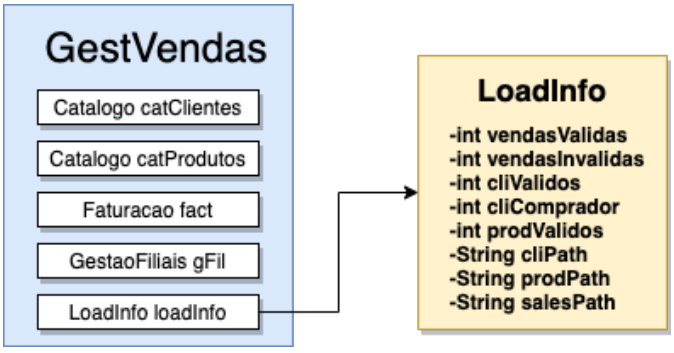
Cada código de Cliente que comprou numa Filial é representado por uma key no mapCli(HashMap) à qual corresponde as informações sobre as suas compras(CliInfo).

A classe ProdInfo guarda para cada mês códigos dos Clientes/Produtos ao qual corresponde um ProdCliinfo que guarda o mês, a faturação e as unidades relativas à venda.

A classe CliInfo é uma extensão da classe ProdInfo que também guarda o número de compras e o gasto total em cada mês para o Cliente.

A classe Filial guarda os Clientes compradores da Filial e o Clientes compradores e as vendas de cada mês da mesma Filial.

3.4 GestVendas



A classe GestVendas é a classe agregadora de todo o projeto, contém um Catalogo para Produtos, outro para Clientes e instâncias das classes Faturacao, GestaoFiliais e LoadInfo.

A classe LoadInfo guarda informações estatísticas relativas ao carregamento dos ficheiros.

4 Interação com o Utilizador

```
-----------------------------------------------------
                      INÍCIO
-----------------------------------------------------
1 | Consultas estatisticas
2 | Consultas interativas
3 | Gravar estado
4 | Carregar estado a partir de um ficheiro
5 | Carregar dados a partir do ficheiro de vendas
0 | Sair
-----------------------------------------------------
Introduza um número:
```

Quando o utilizador corre o programa, é-lhe apresentado o menu com os comandos possíveis de utilizar. Os comandos 1 e 2 apenas podem ser executados após a leitura dos ficheiros, feita através do comando 4 ou 5.

```
-----------------------------------------------------
               CONSULTAS ESTATÍSTICAS
-----------------------------------------------------
1 | Dados relativos ao último ficheiro de vendas lido
2 | Total compras por mês
3 | Facturação total e por (Filial/Mês)
4 | Clientes compradores distintos por (Filial/Mês)
0 | Voltar
-----------------------------------------------------
Introduza um número:
```

```
-------------------------------------------------------------------------------------
                             CONSULTAS INTERATIVAS
---+---------------------------------------------------------------------------------
 1 | Códigos Produtos nunca comprados e total
 2 | Total vendas e clientes distintos para cada filial e para um dado mês
 3 | Número compras, produtos distintos e gasto total de um dado cliente para cada mês
 4 | Número compras, clientes distintos e faturação total de um dado produto para cada mês
 5 | Lista produtos que um cliente mais comprou(quantidade)
 6 | X produtos mais vendidos em todo o ano(unidades) e clientes distintos que os compraram
 7 | 3 maiores compradores em termos de faturação por filial
 8 | X clientes que compraram mais produtos diferentes
 9 | X clientes que mais compraram um produto e valor gasto para cada um
10 | Faturação total de cada produto mês a mês e filial a filial
 0 | Voltar
---+---------------------------------------------------------------------------------
Introduza um número:
```

Após executar o comando 1 ou 2, será apresentado o Menu das Consultas Estatísticas ou Interativas, respetivamente, onde este poderá executar a ação desejada.

```
-----------------------------------------------------
                      QUERIE 10
-----------------------------------------------------
1 | Ver tabela de faturação de um produto
0 | Voltar
-----------------------------------------------------
Introduza um número:
```

Além disso, foi também criado um menu para a utilização da querie 10, para que a navegação dos resultados desta querie fosse o mais simples possível.

```
-------------------------------------------------------------------------------------
                                Página (1/547)
-------------------------------------------------------------------------------------
L1465   88,00  |  T3219   86,00  |  I3987   82,00  |  R1262   82,00  |  F3174   80,00
R1926   80,00  |  A1576   79,00  |  A3254   79,00  |  B1681   79,00  |  E1323   79,00
H4409   79,00  |  O3060   79,00  |  Q2523   79,00  |  R1959   79,00  |  Z3709   79,00
C1122   78,00  |  H1454   78,00  |  H3212   78,00  |  O3076   78,00  |  Q4870   78,00
T4607   78,00  |  V3547   78,00  |  W3112   78,00  |  X2263   78,00  |  C3864   77,00
C4307   77,00  |  H2908   77,00  |  J3659   77,00  |  K1806   77,00  |  N3201   77,00
-------------------------------------------------------------------------------------
     [N] Next | [P] Previous | [F] First | [L] Last | [#] Page Number | [Q] Quit
-------------------------------------------------------------------------------------
```

Por último, se o utilizador executar uma opção que devolve uma coleção, deparar-se-á com o menu de páginas, onde pode utilizar os comandos da barra para navegar nestas. Introduzindo um número de página, se esta for válida, o menu irá para a página exata.

5 Estrutura do Projeto e Diagrama de Classes

Após acabado o projeto em C, decidimos que um ponto a melhorar no nossa trabalho seria uma melhor organização do modelo MVC. Para uma melhor gestão das classes, decidimos criar três packages (Model, View e Controler).

O Modelo do programa é gerido pela classe GestVendas. Esta classe contém as estruturas do programa, e por sua vez utiliza as classes Catalogo, Faturacao e GestaoFiliais. Além disso, esta classe é responsável pelas Consultas Estatísticas e Interativas do nosso programa.

A Vista do programa é controlada pela classe Apresentacao. Esta classe contém todas as funções que devolvem resultados visuais ao utilizador, bem como funções que devolvem mensagens em forma de String para mais tarde poderem ser utilizadas pelo método printMessage.

O Controlador do programa é gerido pelo módulo Interpretador. Este interage com o utilizador para receber input e, por sua vez, utilizar a classe GestVendas para utilizar dados das estruturas e a classe Apresentação para devolver os resultados visuais.

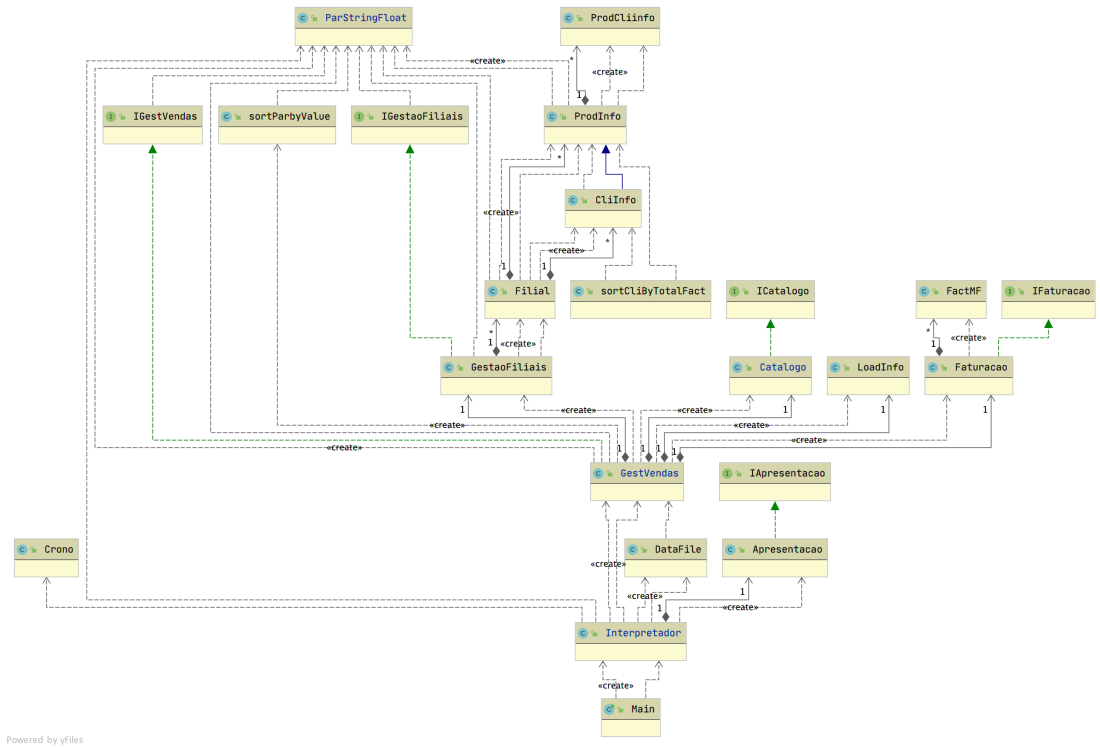


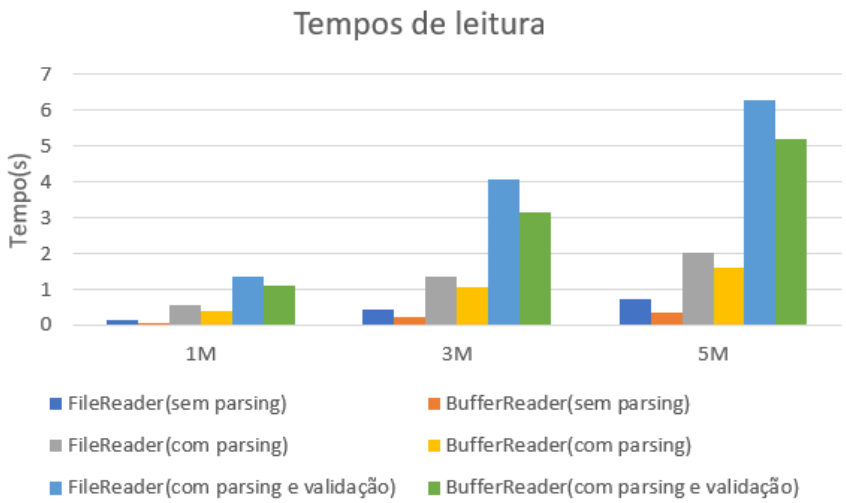
Diagrama de Classes

6 Testes de Performance

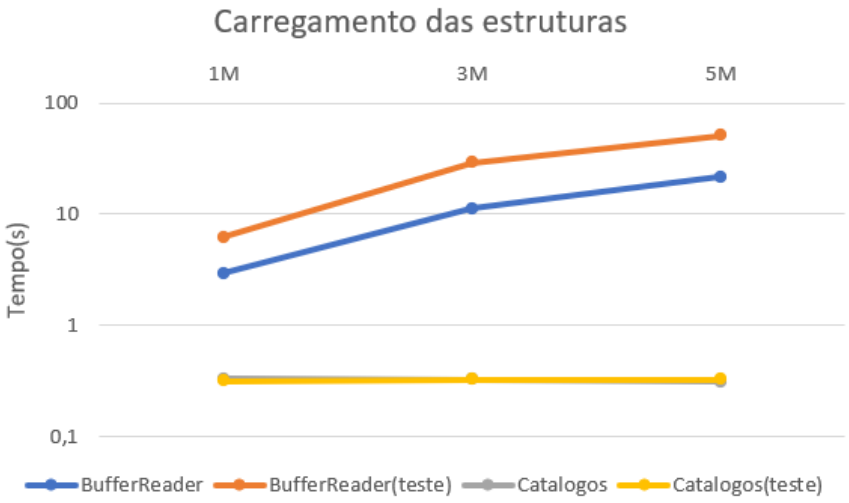
6.1 Tempos de Execução

Posteriormente ao desenvolvimento e codificação de todo o projeto, foi-nos proposto realizar testes de performance que consistiam na medição de vários tempos de desempenho com o objetivo de encontrar a melhor solução. Fizemos medições usando BufferedReader e FileReader e testamos também as queries fazendo alterações nas estruturas(trocamos HashMaps por TreeMaps e TreeSets por HashSets). Cada tempo apresentado é uma média de 20 medições, o computador utilizado tem um i7 de 6 cores com 2.6GHz e 16GB de memória RAM.

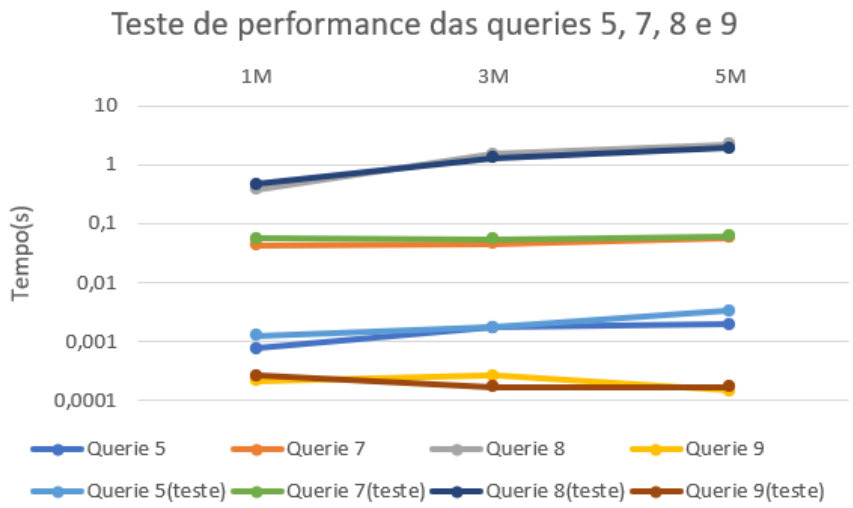
Em seguida, serão demonstrados e explicados os resultados dos mesmos:



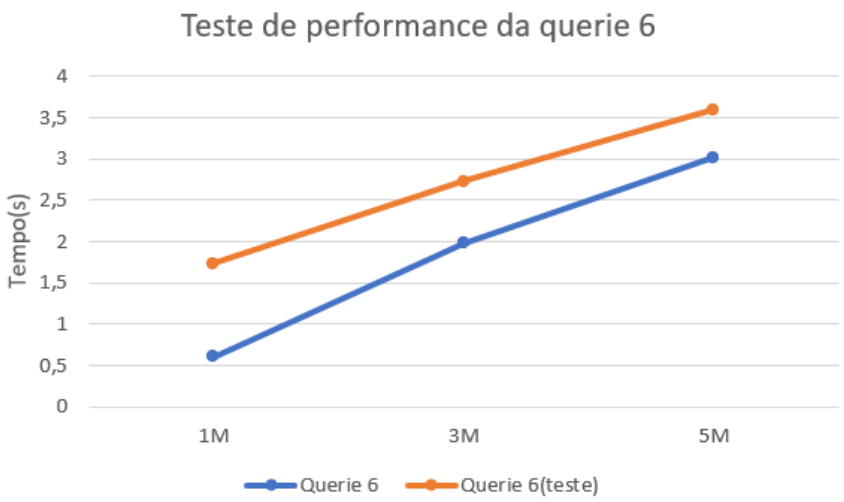
Como podemos ver a partir do gráfico acima, os tempos quer do FileReader quer do BufferReader variam de forma similar, no entanto podemos concluir que o BufferedReader tem uma melhor performance, uma vez que o FileReader precisa de alocar espaço para todas as linhas lidas para as percorrer depois, enquanto que o BufferedReader aloca espaço para apenas uma linha processando-a de imediato, ou seja, o FileReader ocupa muito mais espaço na sua execução do que o BufferedReader. Logo, concluimos que seria melhor utilizar o BufferedReader.



Como podemos observar no gráfico acima, o tempo de preenchimento dos catálogos é constante, enquanto que o preenchimento de todas as outras estruturas tem um tempo superior aquando da realização dos testes, onde trocamos HashMap por TreeMap e TreeSet por HashSet. O tempo de carregamento dos Catalogos é constante uma vez que os ficheiros usados para carregar esta estrutura são sempre os mesmos(Produtos.txt e Clientes.txt). No entanto, seria de prever que o tempo de carregamento aumentasse devido ao tempo de inserção num TreeMap ser superior ao do HashMap, isto não é notado, pois o número de produtos/clientes é reduzido. Por isto optámos por usar um TreeMap nos Catalogos, uma vez que tinha a mesma performance que um HashMap mas os elementos estavam ordenados. Já no caso do carregamento das outras estruturas, como o número de elementos a inserir nas estruturas é muito superior, a diferença do tempo de inserção num TreeMap vs HashMap é notável chegando o tempo de carregamento de 3M de vendas, para os testes, a ser superior ao carregamento de 5M de vendas para as estruturas anteriores. Optamos então por utilizar HashMap nas estrutras à exceção dos Catalogos.

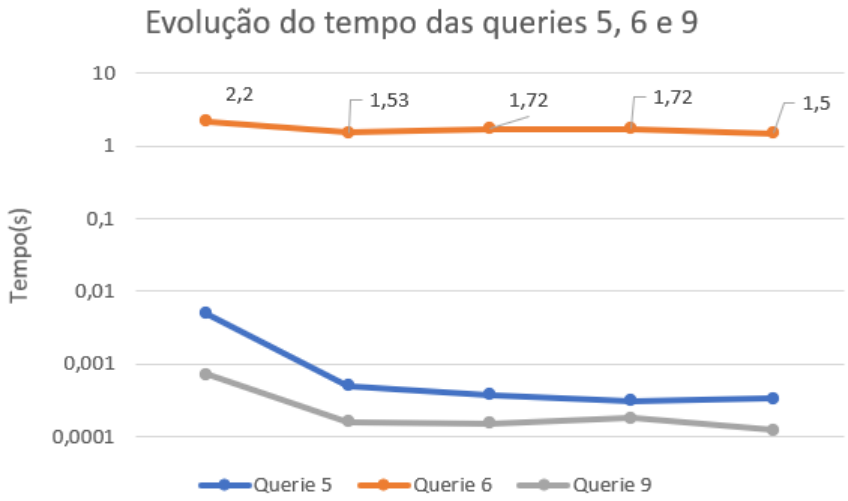


O Cliente usado na medição do tempo de execução da querie 5 foi o Z5000, limite 100000(cem mil) para a querie 8 e o Produto ZZ1999 e o limite 20 para a medição da querie 9. Depois de realizados os testes(alteração de HashMap para TreeMap, TreeSet para HashSet, etc.) para as queries 5, 7 e 9(queries com tempo relativamente pequeno) verificamos através do gráfico acima que a sua performance é similar, pois, apesar de o tempo de procura num TreeMap ser $\mathcal{O}(log(n))$ enquanto que num HashMap é constante, apenas procuramos um Produto/Cliente uma vez no Map que contém todos os Produtos/Clientes(queries 5 e 9) ou consultamos todos os Produtos/Clientes, ou seja, percorremos todas as entradas do Map(querie 7), logo é indiferente ser um HashMap ou um TreeMap. No que toca à querie 8, é calculado o número de produtos diferentes para todos os clientes do catálogo, pelo que o número de procuras no mapCli de cada Filial é cerca de 20000(tamanho do TreeMap), ou seja, cerca de 60000 no total, o que não é um número de procuras suficientemente grande para que haja uma diferença significativa entre o tempo de procura num TreeMap($\prime(log(n))$) e o tempo de procura num HashMap(constante).



O limite usado na medição da querie 6 foi 100000(cem mil). Contrariamente ao observado nas queries anteriores, verificamos que há uma variação significativa na performance destas queries depois da troca das estruturas.

Em relação à querie 6, é feita uma procura de todos os produtos para calcular o número de unidades(feita na classe Faturacao) e, posteriormente, calculamos o número dos diferentes Clientes(dependente do limite introduzido é feita uma procura dos produtos na classe Filial). Uma vez que são feitas cerca de 170000 procuras de produtos na classe Faturacao mais 300000 procuras(100000 por cada filial), que comparando com a querie 8 tem um número de procuras cerca de 8x maior e um tamanho do map cerca de 10x maior, logo é justificável o facto de o tempo de execução aumentar depois dos testes, pois o HashMap tem um melhor tempo de procura.



Ao testar as queries 5, 6 e 9 observamos(tal como está no gráfico acima) que a primeira vez que a querie era executada tinha um significativamente superior em comparação com as execuções seguintes. Para as queries 5 e 9 verificou-se uma queda de tempo de cerca de 90%, enquanto que na querie 6 a queda foi na casa dos 30%. Atribuimos a causa à permanência dos dados em cache, o que permitiu um acesso muito mais rápido a estes em execuções consecutivas, baixando assim o tempo de execução das queries nas vezes seguintes.

7 Conclusão e Reflexão Crítica

Nesta parte do projeto demos foco ao melhoramento das falhas apontadas do projeto em C e à gestão da memória de forma eficiente sem perder performance. Primeiramente, concentrámo-nos em encapsular as classes de forma adequada, clonando as estruturas a serem devolvidas. Além disso, procuramos implementar uma melhor independência da apresentação, fazendo todos os prints nesta e todos os inputs no controlador(à exceção dos inputs dos menus de páginas, que correspondem à apresentação).

Desde logo, tal como no projeto em C, obtivemos resultados positivos no que toca à resposta das queries. De um modo geral, estamos satisfeitos com a nossa solução, tanto a nível de desempenho do programa, como a nível de interação com o utilizador.

No entanto, tínhamos em mente a implementação de uma aplicação em janela, que não conseguimos concretizar. Para contornar a situação, decidimos que, em vez de um único menu, tal como na parte em C do projeto, iriamos criar vários de maneira a tornar a interação com o utilizador mais simples.

Para concluir, os objetivos deste projeto foram atingidos e durante a sua realização foram de reter determinados pontos. Tais como, a necessidade de abdicar de performance para ter um código seguro, protegido do utilizador, como também a importância da organização das estruturas de dados, quais os algoritmos a implementar e como diferentes relações entre estes dois conceitos tiveram um efeito considerável nos tempos de execução das funcionalidades do programa.