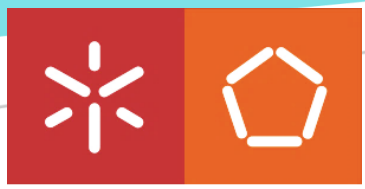


Universidade do Minho
Escola de Engenharia

LABORATÓRIOS DE INFORMÁTICA III

RELATÓRIO PROJETO C

SISTEMA DE GESTÃO DE VENDAS - SGV

Grupo 67

Ana Luísa Carneiro A89533
Ana Rita Peixoto A89612
Luís Miguel Pinto A89506

ÍNDICE

INTRODUÇÃO

O projeto “Sistema de Gestão de Vendas” propõe uma implementação de um programa que seja capaz de processar grandes volumes de dados de maneira eficiente, utilizando os princípios de encapsulamento e modularidade.

Assim, no decorrer do desenvolvimento deste projeto, foram surgindo alguns desafios particulares a cada módulo:

-> O primeiro grande desafio ocorreu na leitura dos ficheiros. Neste ponto, teríamos que descobrir uma solução que se mostrasse eficiente na inserção de cada produto/cliente/venda na estrutura respetiva. Além disso, também teria que ser uma solução eficiente na consulta dos dados.

-> Consequentemente, dado que escolhemos utilizar as implementações do GLib, fomos confrontados com a necessidade de compreender o seu funcionamento e de conhecer as diversas alternativas, de forma a maximizar os seus recursos.

-> O terceiro desafio foi conseguir implementar uma maneira eficiente de apresentar, ao utilizador, um grande volume de resultados. Deste modo, criamos um conjunto de módulos cujo propósito foi permitir ao utilizador navegar pelos resultados “página a página”.

-> Uma outra dificuldade foi desenvolver o nosso programa sem perdas de memória, utilizando como auxílio a ferramenta Valgrind.

-> Por último, achamos que foi desafiante compreender e incluir no nosso projeto os conceitos de encapsulamento e modularidade, dado a se tratar de princípios novos aos quais nos tivemos de adaptar.

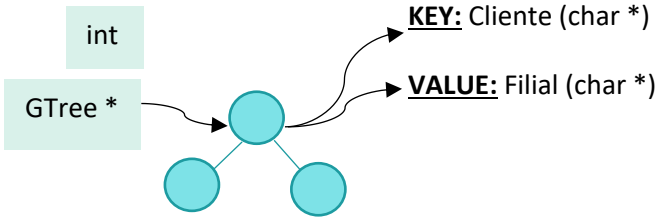
ESTRUTURA DE DADOS

Inicialmente, efetuamos a leitura de cada ficheiro para um *array* estático. O método de inserção era a ordem de leitura. No final, com o auxílio da função *qsort*, colocavamos o array ordenado alfabeticamente. Esta solução era inadequada. Ao utilizar *array*s estáticos, poderíamos estar a desperdiçar memória, ou por outro lado, não ter memória suficiente para todos os dados. Assim, avançamos para a implementação com *arrays* dinâmicos, em que poderíamos controlar o tamanho do mesmo, e deste modo, alocar apenas o espaço necessário. Depois desta implementação para o Catálogo de Produtos e Clientes, decidimos implementar o mesmo método na leitura do ficheiro "Vendas_1M.txt", priviligiando a procura binária($\theta(\log_2 N)$) dos clientes e produtos(no catálogo respetivo) presentes nas vendas, de modo a verificar a validade de cada venda. Ora, neste ponto, o tempo de execução estava aceitável na medida em que demorava apenas alguns segundos. No entanto, consideramos que podíamos melhorar ainda mais e criar estruturas mais sofisticadas de modo que pudessem suportar os objetivos presentes em cada query. Assim, decidimos utilizar a biblioteca GLIB, implementando árvores de procura binária balanceadas em diversos pontos e estruturas do nosso projeto.

A utilização de árvores, ao contrário dos *arrays* de *string*, permite adicionar informação extra a cada *key* presente na árvore, utilizando o campo *value*. Permite também a ordenação da árvore de acordo com uma função de ordenação, no momento de inserção dos dados.

Catálogo de Clientes

A estrutura implementada para o **Catálogo de Clientes** poderá ser representada através da seguinte representação esquemática:

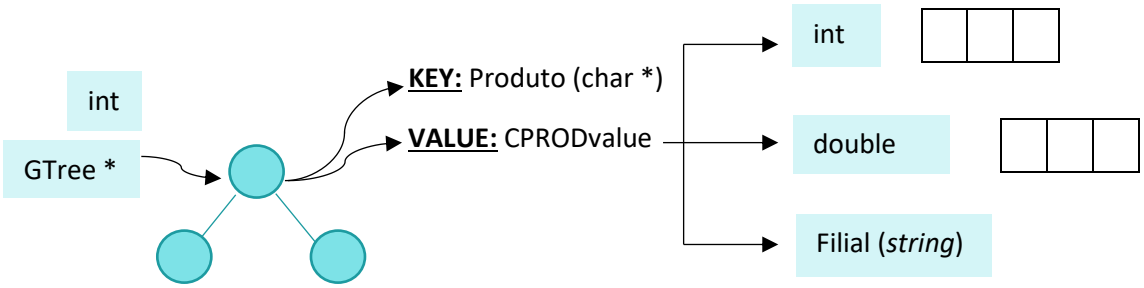


Esta estrutura contém um valor do tipo inteiro, cuja função é armazenar o número de clientes que foram lidos, e também uma árvore binária, onde cada *key* corresponde a um cliente, e cada *value* de cada cliente contém a concatenação numa *string* das filiais onde esse cliente efetuou compras. A função (GCompareDataFunc) utilizada para esta GTree, foi a própria *strcmp* para que os clientes ficassem ordenados por ordem alfabética.

Catálogo de Produtos

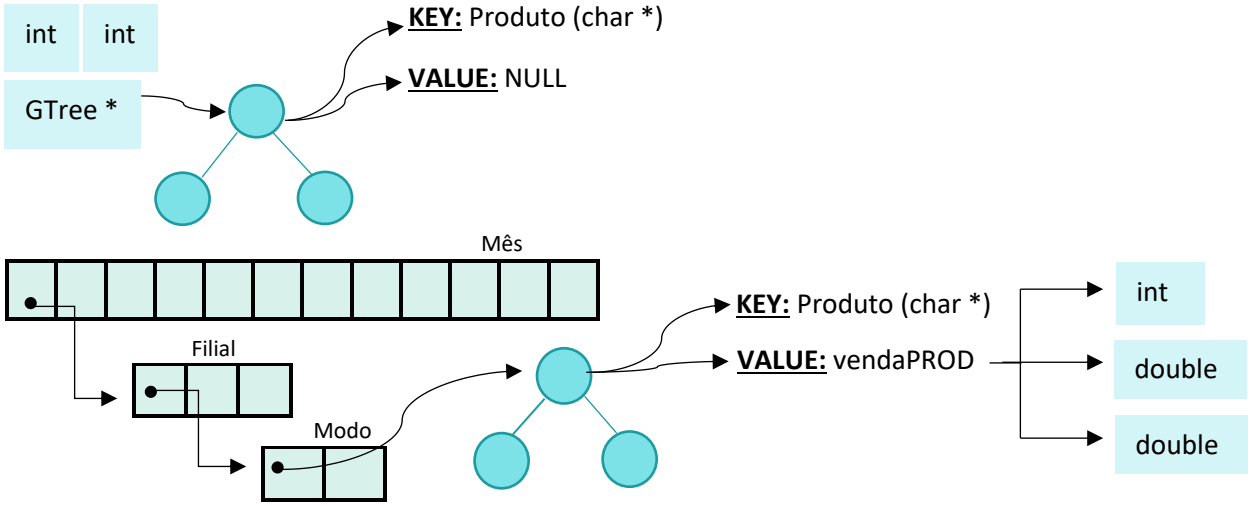
No caso da estrutura para o **Catálogo de Produtos**, a opção foi semelhante. Utilizamos um inteiro que contém o número de produtos lidos, e uma árvore binária (cuja função de comparação utilizada foi *strcmp*, de modo a organizar os produtos por ordem alfabética) em que cada *key* corresponde a um produto. O *value* de cada produto, corresponde a uma estrutura (CPRODvalue), que contém 2 *arrays* estáticos com 3 entradas cada um, de modo que seja

possível armazenar valores filial a filial. Deste modo, criamos um *array* de inteiros cujo conteúdo será o número de clientes que compraram o produto na filial respetiva, e um *array* de *double* que armazenará a quantidade desse produto que foi comprada, filial a filial. Assim, tem-se:



Faturação global

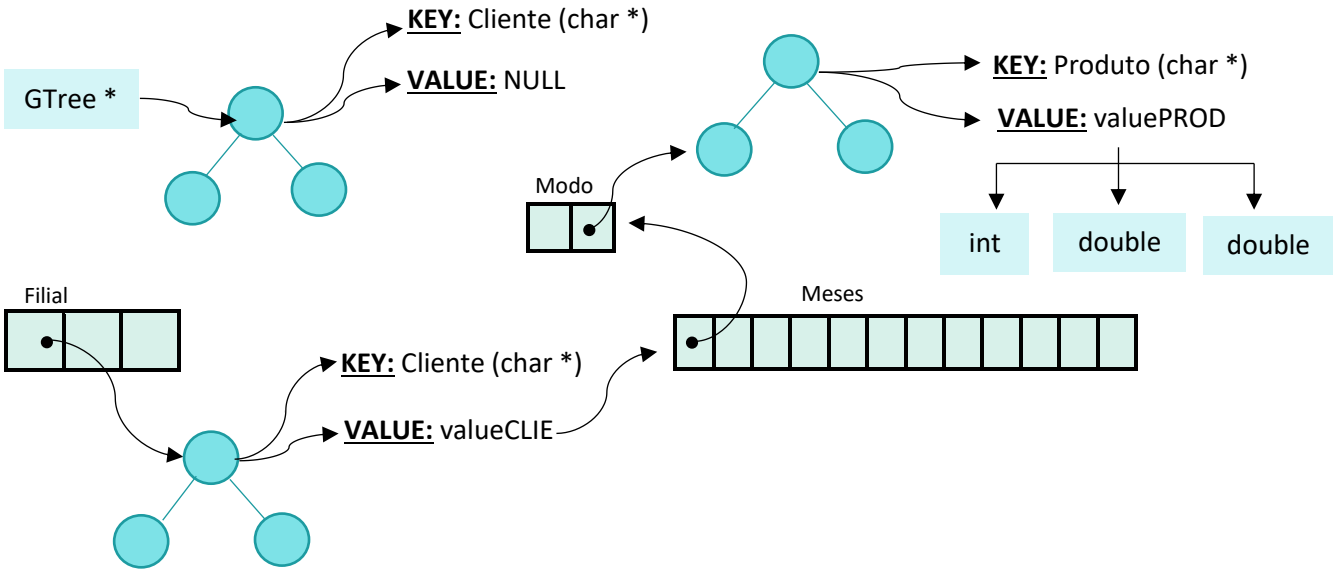
Para a **Faturação Global**, o objetivo era encontrar uma solução que relacionasse cada produto às suas vendas mensais, distinguindo-os em modo N(normal) ou P(promoção). Esquematicamente, a opção do grupo para a implementação desta estrutura, traduz-se em:



Assim, esta estrutura contém 2 valores do tipo inteiro, cuja função é armazenar o número de vendas lidas e válidas. Contém também uma árvore binária, que armazena os produtos que não foram vendidos, utilizando como função de comparação a função *strcmp*. Por último, para completar o objetivo proposto, decidimos implementar um *array* com 12 entradas (para que seja possível distribuir os produtos pelos 12 meses), em que cada uma tem um *array* com 3 entradas (de modo a dividir os produtos filial a filial), onde cada filial tem um *array* com 2 entradas (onde se divide os produtos entre modo normal ou promoção). Neste último *array*, implementamos uma árvore em cada uma das suas 2 entradas, em que a *key* corresponde aos produtos que serão inseridos por ordem alfabética (com o auxílio da função *strcmp*), e o *value* será uma estrutura (vendaPROD) que contém informações relativas àquele produto, tais como a faturação associada, a quantidade vendida e o número de vendas.

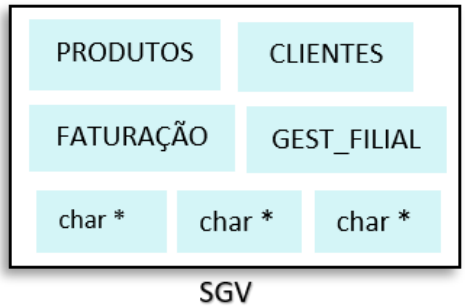
Gestão de Filial

Em relação à **Gestão de Filial**, o esperado seria uma estrutura que relacionasse produtos e clientes, de modo que fosse possível saber, para cada produto, informações como: clientes que o compraram, as unidades que cada um comprou, etc. Assim, de modo a responder a estes objetivos, a estrutura implementada baseia-se na seguinte esquematização:



Deste modo, a estrutura para a Gestão de Filial contém uma árvore onde são inseridos os clientes que não efetuaram compras, por ordem alfabética (com o auxílio da função *strcmp*). Além disso, a estrutura deste módulo contém também um *array* com 3 entradas (de modo a dividir os clientes por filial), em que cada uma dessas entradas contém uma GTree, que organizará os clientes por ordem alfabética, tendo na sua *key* cada um dos clientes. O *value* de cada cliente é uma outra estrutura (valueCLIE) que contém um *array* com 12 entradas (para cada um dos 12 meses), onde cada uma tem um *array* com 2 entradas (de modo a ser possível dividir os produtos em modo normal e em promoção). Neste último *array*, existe uma árvore em cada uma das suas entradas, de modo a organizar os produtos. Assim, a *key* desta árvore corresponde a um produto, e o seu *value* respetivo corresponde a uma estrutura (valuePROD) que contém informações importantes acerca de cada produto, tais como a sua faturação, a quantidade e o número de vendas.

SGV



SGV

Assim, em consequência do referido anteriormente, surge o módulo **SGV** que agrega todas as estruturas anteriormente expostas, de modo a completar a estrutura principal para o Sistema de Gestão de Vendas proposto a desenvolver. Tem-se:

De acordo com a representação esquemática, a estrutura SGV contém as diversas estruturas onde estão armazenados os dados lidos dos ficheiros, e também 3 valores do tipo *char** onde são guardados os caminhos/nomes dos ficheiros, caso o utilizador não queira ler ficheiros pré-definido.

ARQUITETURA DA APLICAÇÃO

Face ao enunciado do projeto, desenvolvemos os seguintes módulos:

main: responsável pela sincronização entre utilizador e programa, i.e, controlador principal do fluxo do programa.

view: responsável pela apresentação da situação atual, da interação com o utilizador, no ecrã.

interface: contém o conjunto de funcionalidades suportadas pela aplicação (as 13 queries do SGV).

leitura: responsável pela leitura dos dados (ficheiros clientes, produtos e vendas) e redireccionamento de forma a serem devidamente tratados.

queries: contem estruturas de dados auxiliares para a implementação de queries (3, 6, 7, 8 e 11).

gestFilial: organiza os dados conforme a filial, apesar de vocacionado para os clientes faz também a ligação do cliente com os respetivos produtos.

catClientes: contem a estrutura com o catálogo dos clientes e respetivas funções sobre clientes.

factGlobal: orientada para produtos, lida com as faturações, em modo: Normal ou Promoção.

catProd: contem a estrutura com o catálogo de produtos e respetivas funções sobre produtos.

Com o efeito de oferecer uma melhor experiência ao utilizador, foi inicialmente desenvolvido o modulo navegador, contudo devido a sua complexidade e de forma a cumprir os requisitos do projeto, este teve de ser dividido em três módulos:

navController: "mini-controlador-de-fluxo", após invocado pelo controlador *main* tem temporariamente o controlo (enquanto o utilizador navega pelos resultados), ou seja, simplesmente avança ou recua páginas da apresentação de resultados.

navModel: modelo de dados específico para o conjunto de navegador.

navView: "vista secundária" do programa, apresenta no ecrã páginas e páginas com resultados, podendo estes ser *ranks*, produtos ou clientes.

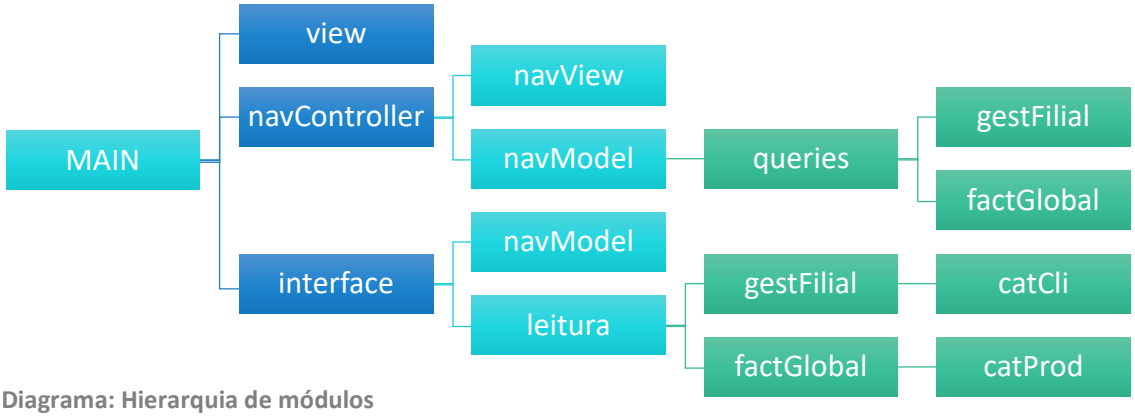
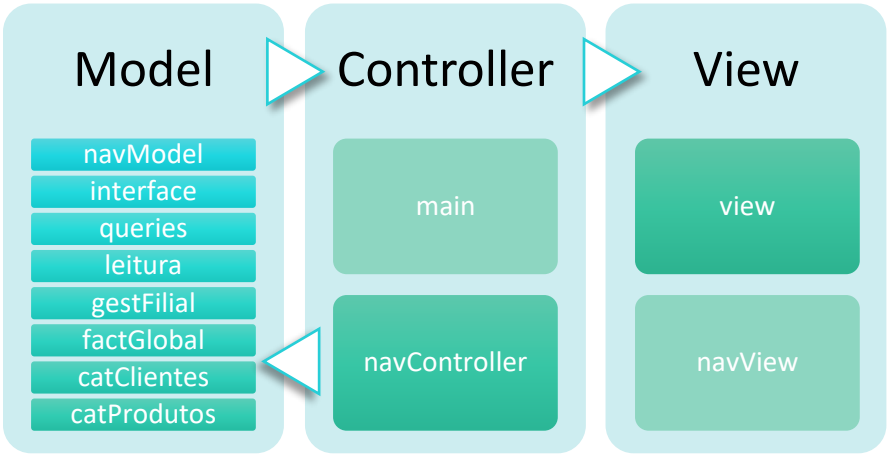


Diagrama: Hierarquia de módulos

MVC – Model, View, Controller

Toda a aplicação do programa foi construída com base no modelo MVC. A divisão de cada módulo pelas três diferentes categorias (Model – Camada de dados e algoritmos, View – Apresentação ao utilizador e Controller – Controlo de fluxo) é dada no diagrama abaixo.



TESTES DE PERFORMANCE

Os testes de performance foram realizados através da média de 3 medições com a introdução dos mesmos dados em todos os ficheiros testados:

Leitura: ---
Query6: ---
Query7: Cliente: L1782
Query8: Mês min.: 1 Mês max.: 12
Query9: Produto: OP1244 Filial: 2
Query10: Cliente: L1782 Mês: 2
Query11: Número: 12
Query12: Cliente: L1782 Número: 12

Para o cálculo do tempo de execução foi usada a biblioteca **time.h** e a função **clock()** dessa API que retorna o tempo decorrido desde que se iniciou a execução. A partir desse cálculo determinou-se a seguinte tabela[1]:

FICHEIROS		Leitura	Query6	Query7	Query8	Query9	Query10	Query11	Query12
Vendas 1M	Trial 1	8,36	$3,6 * 10^{-5}$	$3,1 * 10^{-4}$	0,22	0,062	$9,7 * 10^{-4}$	0,22	$6,0 * 10^{-4}$
	Trial 2	9,87	$7,0 * 10^{-6}$	$9,5 * 10^{-5}$	0,27	0,059	$2,9 * 10^{-4}$	0,26	$3,5 * 10^{-4}$
	Trial 3	10,0	$6,0 * 10^{-6}$	$4,3 * 10^{-5}$	0,27	0,063	$7,8 * 10^{-5}$	0,26	$6,7 * 10^{-4}$
	Media	**9,41 s**	$\mathbf{1,6 * 10^{-5}}$	$\mathbf{2,8 * 10^{-4}}$	**0,25**	**0,06**	$\mathbf{1,6 * 10^{-4}}$	**0,25**	$\mathbf{5,4 * 10^{-4}}$
Vendas 3M	Trial 1	29,41	$2,8 * 10^{-5}$	$3,9 * 10^{-4}$	0,85	0,12	$3,5 * 10^{-4}$	0,25	$2,2 * 10^{-3}$
	Trial 2	35,97	$2,0 * 10^{-5}$	$4,8 * 10^{-4}$	0,83	0,12	$3,0 * 10^{-4}$	0,28	$3,3 * 10^{-2}$
	Trial 3	36,33	$2,7 * 10^{-5}$	$4,2 * 10^{-4}$	0,87	0,11	$1,0 * 10^{-4}$	0,28	$3,3 * 10^{-3}$
	Media	**33,90**	$\mathbf{1,9 * 10^{-5}}$	$\mathbf{4,3 * 10^{-4}}$	**0,85**	**0,12**	$\mathbf{2,5 * 10^{-4}}$	**0,27**	$\mathbf{1,3 * 10^{-2}}$
Vendas 5M	Trial 1	57,23	$2,6 * 10^{-5}$	$4,0 * 10^{-4}$	1,51	0,139	$2,8 * 10^{-3}$	0,24	$2,8 * 10^{-3}$
	Trial 2	65,61	$2,0 * 10^{-5}$	$5,7 * 10^{-4}$	1,48	0,138	$1,7 * 10^{-4}$	0,29	$1,8 * 10^{-2}$
	Trial 3	66,39	$3,2 * 10^{-5}$	$5,4 * 10^{-4}$	1,58	0,144	$3,0 * 10^{-3}$	0,28	$5,7 * 10^{-3}$
	Media	**63,08**	$\mathbf{2,0 * 10^{-5}}$	$\mathbf{5,0 * 10^{-4}}$	**1,52**	**0,14**	$\mathbf{2,0 * 10^{-3}}$	**0,27**	$\mathbf{8,5 * 10^{-5}}$

[1] todos os tempos de execução encontram-se em segundos

Interpretação dos Resultados

Leitura: Tal como seria esperado, quanto maior será o número de vendas a serem lidas pelo programa mais este demora a introduzir os dados em cada um dos módulos das vendas.

Query6: Para a query6 era pedido que se determinasse o número de clientes que nunca compraram e os produtos que nunca foram vendidos. Como podemos comprovar pela tabela, a ordem de grandeza é igual em todos os ficheiros uma vez que a divisão dos clientes que nunca compraram assim como os produtos que nunca foram vendidos está a ser feita da *Leitura*, logo o tempo de execução é praticamente igual independentemente do ficheiro lido.

Query7: Para esta query7 pedia-se uma tabela com o número total de produtos comprados por um dado cliente. Comprova-se que nesta query existe diferença na ordem de grandeza do tempo de execução, devido ao diferente número de vendas. Assim para um ficheiro de 1M é mais fácil encontrar um determinado cliente do que nos outros ficheiros. A contabilização do número de produtos comprados em cada mês também faz diminuir a performance.

Query8: Na query8 era pedido para num dado intervalo fechado de meses determinar o total de vendas. Como era esperado o tempo de execução aumenta com o número de vendas, logo a performance diminui.

Query9: Nesta query pede-se para determinar os códigos dos clientes que compraram um determinado produto numa dada filial. Apesar da execução aumentar com o número de vendas, podemos verificar que a execução de 3M de vendas é mais lenta que de 5M. Isto vem do facto de que pode haver códigos de produtos numa determinada filial que se encontram mais vezes no ficheiro de 3M do que no ficheiro de 5M, tal como acontece no com o produto OP1244 na filial 2. Assim o cálculo dos clientes pode ser mais rápido no do 5M do que nos 3M.

Query10: Para a query10 pedia-se para listar os códigos dos produtos que um dado cliente mais comprou num dado mês. Como podemos ver a performance diminui á medida que se aumenta o número de vendas, o que era espectável, contudo tal como acontece com a query9 o tempo de execução depende do número de produtos que o cliente comprou naquele mês. Por exemplo para o ficheiro de 3M existem cerca de 20 produtos para o cliente L1782 no mês 2 e cerca de 8 para o ficheiro 5M. Assim quanto maior for o número de vendas menor será a performance da query.

Query11: Para esta query era necessário determinar o top dos produtos mais vendidos em todo o ano. Pela tabela vemos que o tempo de execução é da mesma ordem de grandeza em todos os ficheiros. Isto acontece uma vez que a pesquisa dos produtos não acontece nas estruturas ligadas às vendas, mas sim no catálogo de produtos que se mantém sempre igual em qualquer um dos ficheiros. Assim a performance desta query é independentemente do ficheiro de vendas.

Query12: Nesta query era pedido o top dos produtos mais comprados por um dado cliente. Pela tabela fica provado que o tempo de execução aumenta com o número de vendas lidas, contudo conseguimos ver pela tabela que para o ficheiro 5M a performance é maior uma vez que o cliente introduzido (L1782) tem mais vendas nos outros dois ficheiros (5M - 7 vendas, 3M – 202 vendas, 1M- 47 vendas). Assim quanto maior for esse número de vendas, mais serão os produtos que o programa terá de percorrer, o que torna a determinação da query mais lenta.

CONCLUSÃO

Dado por concluído o nosso projeto, apresentamos uma reflexão sobre possíveis melhorias assim como os aspetos a valorizar.

Conseguir concluir o projeto com 0% de memoria perdida, juntamente com o facto das queries estarem 100% operacionais e uma interface gráfica visualmente apelativa e funcional, é um aspeto que consideramos positivo e gratificante.

Por outro lado, poderíamos implementar algumas melhorias neste projeto, como por exemplo, não assumir que o número de vendas de um dado produto é equivalente ao número de clientes que o comprou (query11) pois, apesar de serem valores muito poucos discrepantes haverá sempre uma pequena diferença. Para além disso, na implementação do navegador seria mais interativo dar oportunidade ao utilizador de escolher a configuração da "pagina" (Linhas x colunas). Do mesmo modo, conseguiríamos refinar a apresentação final do navegador através de uma adaptação mais especifica á query9. Para rematar, reconhecemos que a implementação escolhida para a estrutura de gestão de filial não se mostrou eficiente em todos os parâmetros, apesar de ser excelente na consulta direta dos dados (aceder ao mês, modo, etc….) não é adequada para travessias ao longo da estrutura.

De um modo geral, o trabalho cumpriu com os requisitos propostos e apesar de haver melhorias, achamos que o balanço é positivo.