Escola de Engenharia
Universidade do Minho

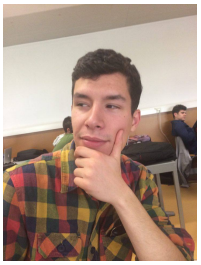
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado Integrado em Engenharia Informática
Laboratórios de Informática III

Gestão de Vendas de uma cadeia de Distribuição com 3 filiais
GEREVENDAS

Grupo 84


Célia Figueiredo
a67637


Gil Gonçalves
a67738


Humberto Vaz
a73236


Ricardo Lopes
a72062

Braga, 26 de Abril de 2016

1. Introdução

No âmbito da unidade curricular de Laboratórios de Informática III do 2ºano da licenciatura de Engenharia Informática foi proposto o desenvolvimento de um projeto em linguagem C que tem por objetivo fundamental ajudar à consolidação dos conteúdos teóricos e práticos e enriquecer os conhecimentos adquiridos nas UCs de Programação Imperativa, de Algoritmos e Complexidade, e da disciplina de Arquitetura de Computadores. Este projeto considera-se um grande desafio para nós pelo facto de passarmos a realizar programação em grande escala, uma vez que se trata de grandes volumes de dados e por isso uma maior complexidade. Nesse sentido, o desenvolvimento deste programa será realizado à luz dos princípios da modularidade (divisão do código fonte em unidades separadas coerentes) e do encapsulamento (garantia de proteção e acessos controlados aos dados).

Conteúdo

2. Descrição dos Módulos

A arquitetura da aplicação a desenvolver é definida por quatro módulos principais: Catálogo de clientes, Catálogo de produtos, Faturação Global e Vendas por Filial, cujas fontes de dados são três ficheiros de texto detalhados abaixo.

No ficheiro **Produtos.txt** cada linha representa o código de um produto vendável no hipermercado, sendo cada código formado por duas letras maiúsculas e 4 dígitos (que representam um inteiro entre 1000 e 1999), como no exemplo:

```
AB9012
XY1185
BC9190
```

O ficheiro de produtos contém cerca de 200.000 códigos de produto.

No ficheiro **Clientes.txt** cada linha representa o código de um cliente identificado no hipermercado, sendo cada código de cliente formado por uma letra maiúscula e 4 dígitos que representam um inteiro entre 1000 e 5000, segue um exemplo:

```
F2916
W1219
F2915
```

O ficheiro de clientes contém cerca de 20.000 códigos de cliente.

O ficheiro **Vendas_1M.txt**, no qual cada linha representa o registo de uma venda efectuada numa qualquer das 3 filiais da Cadeia de Distribuição. Cada linha (a que chamaremos compra ou venda, o que apenas depende do ponto de vista) será formada por um código de produto, um preço unitário decimal (entre 0.0 e 999.99), o número inteiro de unidades compradas (entre 1 e 200), a letra **N** ou **P** conforme tenha sido uma compra **Normal** ou uma compra em **Promoção**, o código do cliente, o mês da compra (1 .. 12) e a filial (de 1 a 3) onde a venda foi realizada, como se pode verificar nos exemplos seguintes:

```
KR1583 77.72 128 P L4891 2 1
QQ1041 536.53 194 P X4054 12 3
OP1244 481.43 67 P Q3869 9 1
JP1982 343.2 168 N T1805 10 2
IZ1636 923.72 193 P T2220 4 2
```

O ficheiro de vendas inicial, **Vendas_1M.txt** , conterá 1.000.000 (1 milhão) de registos de vendas realizadas nas 3 filiais da cadeia de distribuição.

A aplicação possuiu uma arquitectura tal como apresentado na figura seguinte, em que se identificam as fontes de dados, a sua leitura e os módulos de dados a construir:

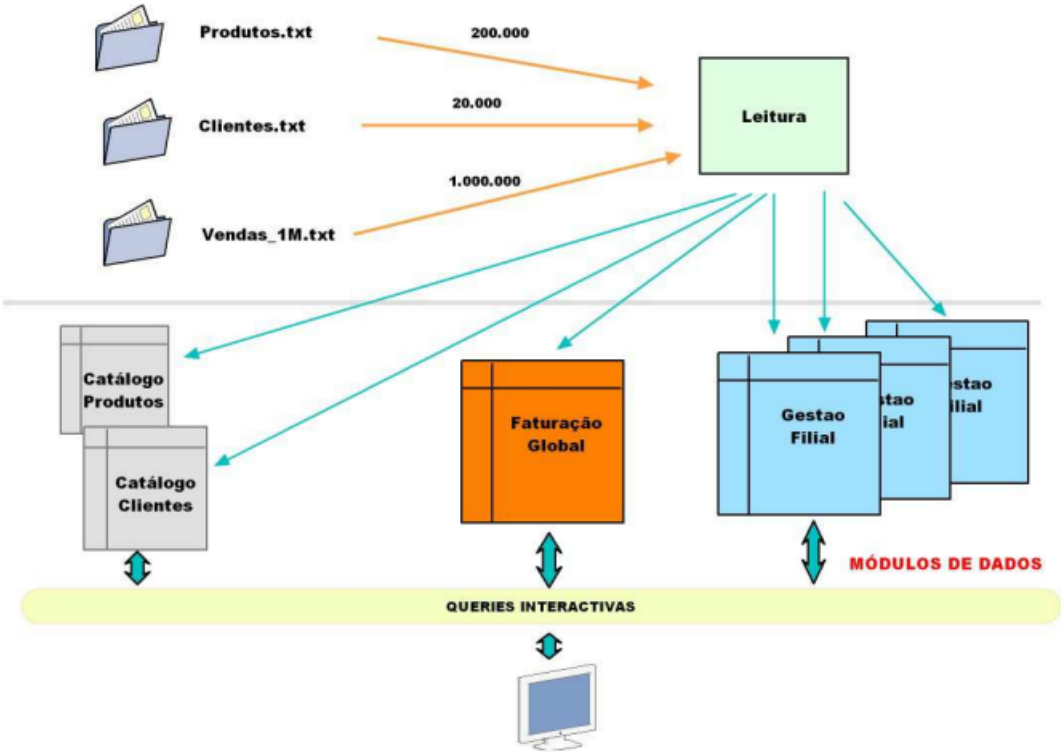


Figura 2.1: Arquitetura da aplicação

2.1 Catálogo de Clientes

É o módulo de dados onde são guardados os códigos de todos os clientes do ficheiro **Clientes.txt**, organizados por índice alfabético;

2.1.1 Clientes.h

Módulo de dados onde são guardados os códigos de todos os clientes do ficheiro **Clientes.txt**. O array de árvores (struct avl_table **array_arv) é um array de 26 posições cujos índices se encontram organizados alfabeticamente. Cada índice contém um apontador para uma árvore correspondente à letra respetiva desse índice.

Tipos Opacos

```
typedef struct catalogo_clientes *CatClientes;
```

/*API*/

- CatClientes inicializa_catalogo_clientes() - Função que cria uma estrutura de clientes vazia;

- void insertC(CatClientes c, char * valor);
- void cat_remove_cliente(CatClientes cat, char *str);
- void free_catalogo_Clientes(CatClientes cat);
- int existeCliente (char *cliente,CatClientes cat);
- int numeroClientes(CatClientes cat);
- int numeroClientesLetra(CatClientes cat, char letra);

2.2 Catálogo de Produtos

Módulo de dados onde são guardados os códigos de todos os produtos do ficheiro **Produtos.txt**, organizados por índice alfabético, o que irá permitir, de forma eficaz, saber quais são os produtos cujos códigos começam por uma dada letra do alfabeto, quantos são

2.2.1 Produtos.h

Tipos Opacos

```
typedef struct catalogo_produtos *CatProdutos;
```

/*API*/

- CatProdutos inicializa_catalogo_produtos();
- void insertP(CatProdutos c, char * valor);
- void cat_remove_produto(CatProdutos cat, char *str);
- void free_catalogo_produtos(CatProdutos cat);
- int existeProduto (char *produto,CatProdutos cat);
- int numeroProdutos(CatProdutos cat);
- int numeroProdutosLetra(CatProdutos cat, char letra);
- ARRAY listaProdutosLetra(CatProdutos cat, char l);

2.3 Faturação Global

Módulo de dados que contém as estruturas de dados responsáveis pela resposta eficiente a questões quantitativas que relacionam os produtos às suas vendas mensais, em modo Normal (N) ou em Promoção (P), para cada um dos casos guardando o número de vendas e o valor total de faturação de cada um destes tipos. Este módulo deve referenciar todos os produtos, mesmo os que nunca foram vendidos. Este módulo não contém qualquer referência a clientes, mas deve ser capaz de distinguir os valores obtidos em cada filial;

2.3.1 faturacao.h

Tipos Opacos

```
typedef struct faturacao *Faturacao;
typedef struct info *Info;
```

/*API*/

- Faturacao inicializa_faturacao();
- void cont_regista_produto(Faturacao fat, char *prod);
- void cont_insere_venda(Faturacao fat, char *produto, int q, float preco, char M,int mes, int filial);
- void cont_remove_produto(Faturacao fat, char *produto);
- void free_faturacao(Faturacao fat);
- int getNumeroTotalVendasNTodasFiliais (char* prod,int mes,Faturacao fat);
- int getNumeroTotalVendasNFilialX (char* prod,int mes,Faturacao fat, int filial);
- int getNumeroTotalVendasPTodasFiliais (char* prod,int mes,Faturacao fat);
- int getNumeroTotalVendasPFilialX (char* prod,int mes,Faturacao fat, int filial);
- float getTotalFatPFilialX (char* prod,int mes,Faturacao fat, int filial);
- float getTotalFatNFilialX (char* prod,int mes,Faturacao fat, int filial);
- int getVendasNFilialX (char* prod,int mes,Faturacao fat, int filial);
- int getVendasPFilialX (char* prod,int mes,Faturacao fat, int filial);
- int getQuantidadeNFilialX (char* prod,int mes,Faturacao fat, int filial);
- int getQuantidadePFilialX (char* prod,int mes,Faturacao fat, int filial);
- ARRAY naoCompradosFilial(Faturacao fat, int filial);
- ARRAY naoComprados(Faturacao fat);
- int totalVendasMeses(Faturacao fat, int a, int b);
- float totalFatMeses(Faturacao fat, int a, int b);
- ARRAY nMaisVendidos(Faturacao fat, int n);
- int getQuantidadeFilial(Faturacao fat, char*prod, int filial);

2.4 Gestão da Filial

Módulo de dados que, a partir dos ficheiros lidos, contém as estruturas de dados adequadas à representação dos relacionamentos, fundamentais para a aplicação, entre produtos e clientes, ou seja, para cada produto, saber quais os clientes que o compraram, quantas unidades cada um comprou, em que mês e em que filial. Para a estruturação optimizada dos dados deste módulo de dados será crucial analisar as queries que a aplicação deverá implementar, tendo sempre em atenção que pretendemos ter o histórico de vendas organizado por filiais para uma melhor análise, não esquecendo que existem 3 filiais nesta cadeia.

2.4.1 Filial.h

Tipos Opacos

```
typedef struct filial *Filial;
typedef struct icliente *Icliente;
typedef struct iprodutos *Iprodutos;
```

- Filial inicializa_filial();
- void fil_regista_cliente(Filial fil, char *cliente);
- void fil_insere_prod(Filial fil, char *cliente, char *produto,int q, int mes, float preco, char p);
- int getQuantidadeMesCliente(Filial fil, char *cliente, int mes);
- ARRAY naoCompraram(Filial fil);
- ARRAY compraram(Filial fil);
- void clientesCompraram(Filial fil,ARRAY a);
- void free_filial(Filial fil);
- ARRAY topMaisGastou(ARRAY a);
- ARRAY clientesCompraramProduto(Filial fil, char* produto);
- int comprouProdutoN(Filial fil, char* cliente, char* produto);
- int comprouProdutoP(Filial fil, char* cliente, char* produto);
- int getNumClientesFilial(Filial fil, char* produto);
- void getIProdMes(Filial fil, char* cliente, int mes, ARRAY a);
- ARRAY extraiPorQuantidade(ARRAY a,int mes);
- void removeNaoCompraram(Filial fil, ARRAY a);
- void removeCompraram(Filial fil, ARRAY a);

3. Main.c

O ficheiro é carregado e de seguida aparece um menu com 12 opções, referentes às 12 queries do projeto, sendo que decidimos usar o [0] para sair do GESTHIPER e o [1] para voltar a ver o menu inicial. O objetivo é que o utilizador prima a tecla correspondente à opção do menu pretendida.

4. Interface do utilizador

Quando o utilizador executa o programa é-lhe pedido que escolha qual o documento de texto que pretende analisar, como podemos observar na figura seguinte:

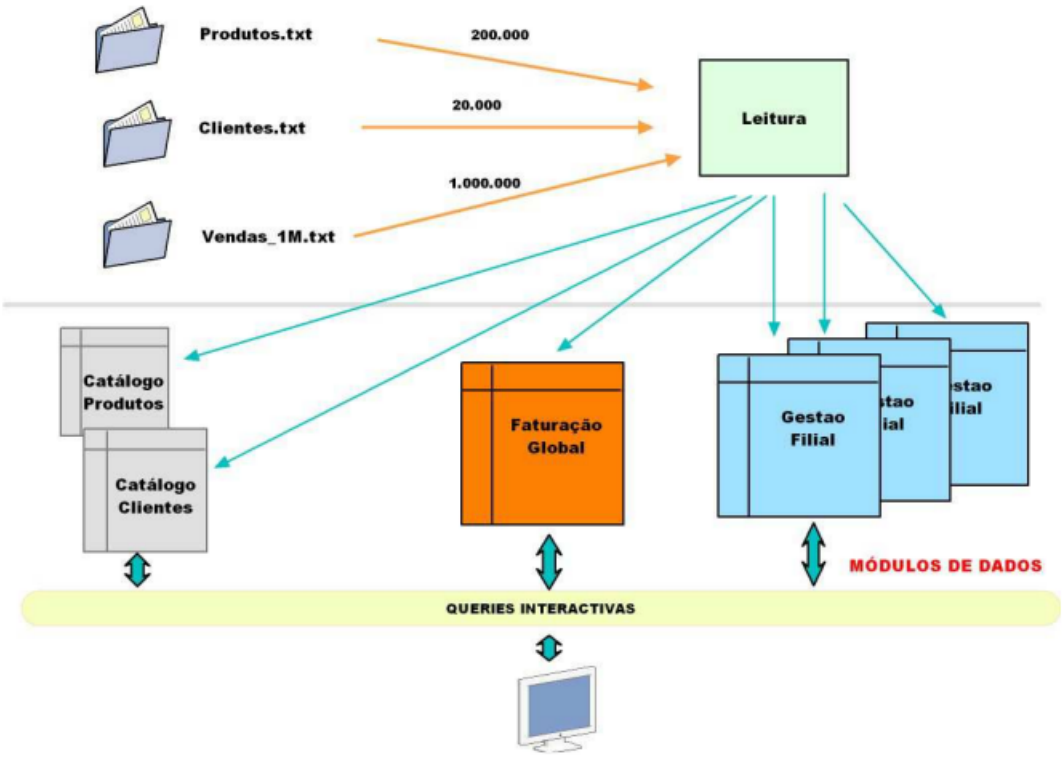


Figura 4.1: Arquitetura da aplicação

5. Resultados e comentários sobre os testes de performance

Depois de desenvolver e codificar todo o projeto foi-nos proposto realizar alguns testes de performance que consistem em comparar os tempos de execução das queries 8, 9, 10, 11 e 12 usando os ficheiros Vendas_1M.txt (1000 000 vendas), Vendas_3M.txt (3 milhões de vendas) e Vendas_5M.txt (5 milhões de vendas). Uma vez que a quantidade de vendas vai aumentando de ficheiro para ficheiro é aceitável que os tempos de execução para os carregar aumente.

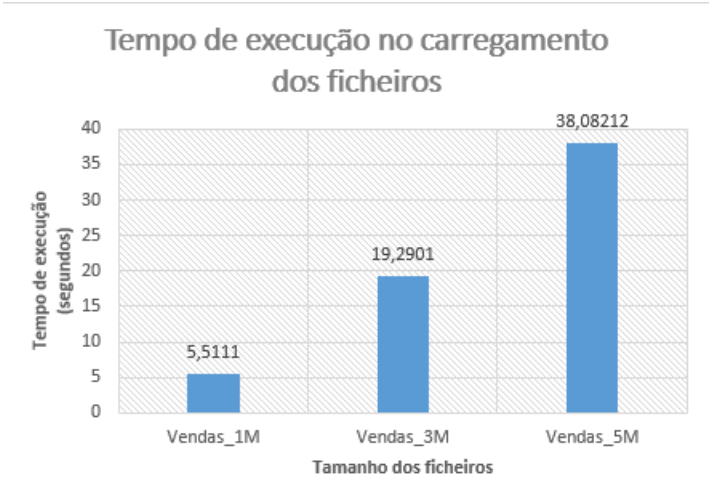


Figura 5.1: Gráfico do tempo de execução do carregamento dos 3 ficheiros de vendas

Comparando os valores de execução das queries pretendidas, como podemos observar nos respetivos gráficos apresentados,

6. Makefile e Grafo de dependências

A makefile permite correr todo o software escrevendo apenas *"make"* no terminal. Posto isto, apresenta-se a makefile utilizada cujas flags utilizadas como opção de compilação são –Wall –Wextra –ansi – pedantic –O2. Possui ainda a opção "make clean" que elimina todos os ".o" que foram criados quando se compilou o software.

```
objects = array.o avl.o clientes.o faturacao.o filial.o \
produtos.o queries.o

CFLAGS=-Wall -ansi -pedantic -O2

all:
make clean
make produtos
make array
make avl
make clientes
make faturacao
make filial
make queries
make leitura

leitura: src/leitura.c array.o avl.o clientes.o faturacao.o filial.o
 produtos.o queries.o
gcc src/leitura.c array.o avl.o clientes.o faturacao.o filial.o
 produtos.o queries.o $(CFLAGS) -o gereVendas -lm

queries: src/queries.c src/headers/queries.h
gcc src/queries.c -c $(CFLAGS)

clientes: src/clientes.c src/headers/clientes.h
gcc src/clientes.c -c $(CFLAGS)

produtos: src/produtos.c src/headers/produtos.h
gcc src/produtos.c -c $(CFLAGS)

array: src/array.c src/headers/array.h
gcc src/array.c -c $(CFLAGS)

faturacao: src/faturacao.c src/headers/faturacao.h
```

```
gcc src/faturacao.c -c $(CFLAGS)

filial: src/filial.c src/headers/filial.h
gcc src/filial.c -c $(CFLAGS)

avl: src/avl.c src/headers/avl.h
gcc src/avl.c -c $(CFLAGS)

.PHONY : clean
clean :
rm -f gereVendas
rm -f $(objects)
rm -f gesval
```

7. Conclusão

Uma vez que se tratou de um trabalho de uma dimensão já considerável comparando com o que estávamos habituados envolveu utilização de técnicas particulares e tivemos sempre como objetivo que este trabalho fosse concebido de modo a que seja facilmente modificável, e seja, apesar da complexidade, o mais optimizado possível a todos os níveis.