



Universidade do Minho

Universidade do Minho

Relatório do Trabalho Prático LI3

Mestrado Integrado em Engenharia Informática

Grupo 31

| | |
|-----------------|--------|
| André Moraes | A83899 |
| Luís Ribeiro | A85954 |
| Pedro Rodrigues | A84783 |

Introdução

O trabalho realizado foi desenvolvido em linguagem C e teve como principal objetivo a gestão das vendas de uma distribuidora com 3 filiais. Para a concretização do objetivo, utilizamos ficheiros “.txt” com informação útil sobre clientes, vendas e produtos. Inicialmente, usamos arrays para extrair a informação dos ficheiros para a manipulação da mesma. Para facilitar a extração de informação dos ficheiros criamos algumas estruturas de dados. A exigência e a demora prolongada do tempo de execução levou ao uso da biblioteca GLib que disponibiliza acesso a estruturas já implementadas, melhorando drasticamente o tempo de execução.

Assim, diminuámos o número de estruturas criadas, tirando o máximo proveito das diferentes estruturas que esta biblioteca disponibiliza.

O objetivo deste relatório é justificar as escolhas dos tipos de dados, estruturas de armazenamento e também explicar as estratégias usadas para a resposta das questões propostas.

Módulos e Estrutura de Ficheiros

Foram criados vários módulos de forma a garantir o encapsulamento. Segue-se uma breve descrição de cada um desses módulos e estruturas de dados:

CatProds

```
typedef struct cat_prods{  
    GHashTable* produtos [26][26];  
    int nr_produtos;  
}cat_prods;
```



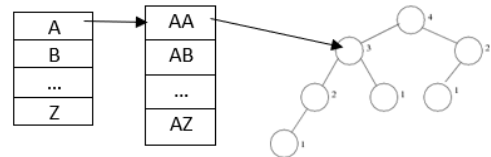
Estrutura de dados

Este módulo serve para guardar todos os produtos válidos lidos do ficheiro “Produtos.txt”.

Neste módulo, inicialmente, foi adotada uma estrutura de arrays, em que foi facilmente detetado que não era a melhor opção devido aos tempos de procura elevados.

Sendo assim adotamos outro método:

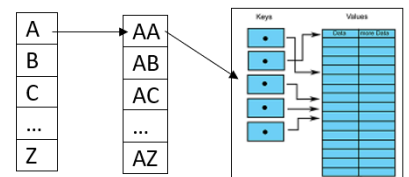
- Um array com 26 posições correspondentes à primeira letra do produto.
- Cada posição desse array aponta para um novo array, novamente com 26 posições, associado à 2ª letra do código do produto.
- Cada posição do segundo array aponta para uma GTree, que contém todos os produtos começados por essas letras.



```
=> Vendas.txt : Das 1000000 vendas lidas, 891108 sao validas
CPU Time: 8.642567
```

Tempo de execução com GTree

Mantivemos esta estrutura durante quase todo o trabalho, mas no final reparamos que poderíamos melhorar a eficiência da leitura de dados e portanto, decidimos utilizar GHashTable em vez de GTree, tendo esta um tempo de procura constante, o que nos fez ganhar 2 segundos relativamente aos 8 segundos anteriores, visto que a maior parte das funções utilizadas por este módulo são apenas de procura.



Estrutura de dados final

Na imagem seguinte, apresentamos a API do módulo:

```
Cat_Prods inicializa_CatProds();
Cat_Prods insereProd (Cat_Prods catp, Produto p);
int existeProd (Cat_Prods catp, Produto p);
int nrProds (Cat_Prods catp);
Lista_Prods produtosPorLetra (Cat_Prods catp, char letra);
GHashTable* getProdutosLetra (Lista_Prods lst, int i);
```

➡ API

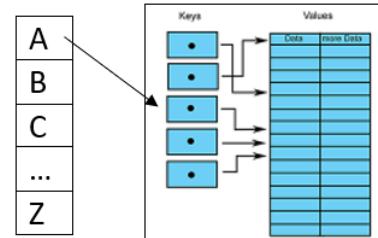
CatCli

```
typedef struct cat_cli{
    GHashTable* clientes [26];
    int nr_clientes;
}cat_cli;
```

➡ Estrutura de dados

Este módulo é idêntico ao anterior e, portanto, utilizamos sempre a mesma estratégia e estruturas de dados. Serve para guardar todos os clientes válidos lidos do ficheiro “Clientes.txt”.

Tal como o módulo acima, foi adotada inicialmente uma estrutura de arrays. Mais tarde um array em que cada posição correspondia à primeira letra do código do cliente e apontava para uma GTree que guardava todos os códigos começados por essa letra. No final, para melhorar o desempenho de procura, alteramos, tal como anteriormente, a GTree por uma GHashTable e as diferenças foram notórias.



Para este módulo definimos a seguinte API:

```
Cat_Cli inicializa_CatCli();
Cat_Cli insereCli(Cat_Cli catc, Cliente c);
int existeCli (Cat_Cli catc, Cliente c);
GHashTable* clientesLetra(Cat_Cli catc, char l);
int nrCli (Cat_Cli catc);
```



Clientes

Este módulo é utilizado para encapsular o cliente. Definimos o cliente como uma string (ou char*) e definimos a seguinte API para que os outros módulos pudessem utilizar a estrutura Clientes:

```
Cliente criaCliente (char* cliente);
char* getCliente (Cliente c);
char primeiraLetraCli (Cliente c);
int validaCliente (Cliente c);
```



Produto

Tal como o anterior, é utilizado para encapsular o produto, sendo este definido como uma string (char *) e foi definida a seguinte API:

```
Produto criaProduto (char * produto);
char* getProduto (Produto p);
char primeiraLetra (Produto p);
char segundaLetra (Produto p);
int validaProduto (Produto p);
```



Venda

Este módulo serve para encapsular o tipo Venda, que contém uma estrutura diferente das outras:

```
typedef struct vend{
    char* produto;
    float preco;
    int unidades;
    char compra;
    char *cliente;
    int mes;
    int filial;
}vend;
```



Estrutura de dados

Perante a requisição dos outros módulos, foi definida a seguinte API:

```
Venda criaVenda (char* venda,Cat_Prods catp,Cat_Cli catc);
int getUniVenda (Venda v);
float getPrecoVenda (Venda v);
char * getProdVenda (Venda v);
int getMes (Venda v);
char getTipoVenda (Venda v);
char * getCliVenda (Venda v);
int getFilial (Venda v);
```



API

Faturação

Este módulo guarda a informação correspondente às vendas, organizadas consoante o produto vendido.

```
typedef struct faturacao{
    /*
     * N 0
     * P 1
     */
    GTree* fat [12][2];
    GArray* nc;
}faturacao;

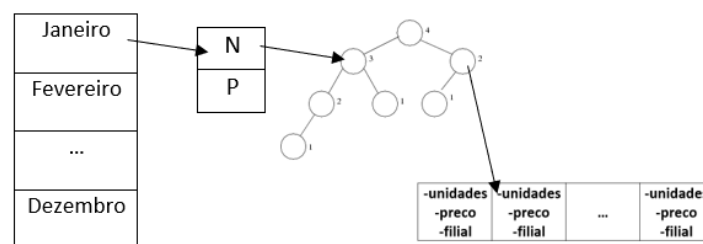
typedef struct fatnode{
    GPtrArray* fatvendas;
}fatnode;

typedef struct fat_str{
    int unidades;
    float preco;
    int filial;
}fat_str;
```



Estrutura de dados

Neste módulo decidimos organizar a informação por meses e tipo de compra. Assim, a estrutura de dados da *Faturacao* é composta por um array com 12 posições corresponde ao mês em que a venda foi feita em que cada posição aponta para um array com 2 posições distinguido assim a venda do tipo normal e em promoção. Cada posição desse array aponta para uma *GTree* onde as vendas são organizadas por produtos, ou seja, a key da *GTree* corresponde ao código do produto vendido e os dados dentro do nodo são guardados num *GPtrArray* que contém uma estrutura com as unidades, o preço e a filial correspondente a essa venda.



Estrutura de dados Final

O *GArray* *nc* presente na estrutura da *Faturacao* corresponde ao código dos produtos que nunca foram vendidos, é atualizado quando é pedida a query correspondente e mantido durante o correr do programa, aumentando assim a execução dessa query caso esta seja corrida mais do que uma vez.

Para este módulo, achamos adequada a seguinte API:

```
Faturacao inicializaFaturacao();
Faturacao insereVenda(Faturacao fatp, Venda v);
int nrVendaProduto(Faturacao fatp, Produto p, int mes, char tipo);
float fatProduto(Faturacao fatp, Produto p, int mes, char tipo);
int atualizanc (Faturacao fatp, Cat_Prods catp);
int getNaoComprados_nr (Faturacao fatp);
GArray* getNaoComprados (Faturacao fatp);
int getVendasEntre(Faturacao fatp, int x, int y);
float getFatEntre(Faturacao fatp, int x, int y);
GArray* getNaoCompFil(Faturacao fatp, Cat_Prods catp, int filial);
```



API

Filial

```
typedef struct filial{
    /*
     * N 0
     * A
     * P 1
     */
    GTree* fil [3][12];
    GArray* compT;
    int nc;
}filial;

typedef struct fnode{
    GPtrArray* filvendas;
}fnode;

typedef struct fi_str{
    char* p;
    char tipo;
    int unidades;
    float preco;
}fi_str;
```



Estrutura de dados

Este módulo tem como objetivo guardar as informações da compra organizadas filial, contendo as informações de cliente e produto.

A sua estrutura de dados é muito semelhante à *Faturacao*, temos um array que aponta para outro array que aponta para uma *GTree* de *GPtrArray*.

Assim sendo, a estrutura é composta por um array de 3 posições que correspondem às filiais e divide as vendas pelas três filiais. Cada uma dessas posições aponta novamente para um array, em que cada posição representa o mês em que a venda foi registada, ou seja, será um array de 12 posições. Cada posição aponta para uma *GTree* onde estão organizadas por código de cliente, ou seja, a key de cada nodo da *GTree* é o código do cliente, dentro de cada nodo temos um *GPtrArray* que contém a informação do produto, do tipo da venda, das unidades e do preço unitário.

A estrutura contém ainda um *GArray* compT que guarda os clientes que compraram em todas as filiais e o inteiro nc que guarda os clientes que não compraram.

Definimos então a seguinte API para tornar possível a execução das queries:

```
Filial inicializaFilial();
Filial insereFilV (Filial f, Venda v);
int nr_clientesCompraram(Filial f, int filial, int mes);
GArray* clientesCompraram(Filial f, int filial, int mes);
int nr_compraCli (Filial f, Cliente c, int mes, int filial);
int fat_compraCli (Filial f, Cliente c, int mes, int filial);
int cliente_nc(Filial f, Cat_Cli catc);
GArray* getCompT(Filial f, Cat_Cli catc);
int vendasProdutoFil(Filial f, Produto p, int mes, char tipo, int fil);
float fatProdutoFil(Filial f, Produto p, int mes, char tipo, int fil);
GArray* cliCompProd (Filial f, Produto p, int filial, char tipo);
GArray* tresProdutos (Filial f, Cliente c);
GArray* prodCompDec (Filial f, Cliente c, int mes);
GArray* prodNcomp (Filial f, int filial, int N);
```



API

Navegador

Neste módulo contemos todas as funções referentes a menus. Inicialmente é chamada no início do programa a função que apresenta o menu inicial, que está ilustrado na figura abaixo, e, consoante a escolha do utilizador, é chamada a função do menu da respetiva query onde se pede ao utilizador os argumentos necessários para a execução da mesma.

```
-----
| Sistema de Faturacao de Vendas |
-----

==> Escolha uma opcao <==

1 -> Ler os ficheiros
2 -> Lista e numero de produtos começados por uma letra
3 -> Numero de vendas e faturacao com um produto em determinado mes
4 -> Lista e numero de produtos que nunca foram comprados
5 -> Clientes que realizaram compras em todas as filiais
6 -> Numero de clientes sem compras e produtos nao comprados
7 -> Tabela com cliente e total comprado em cada mes por filial
8 -> Numero de vendas e total faturado entre dois meses
9 -> Clientes que compraram um produto na filial
A -> Produtos mais comprados pelo cliente naquele mes
B -> Lista com N produtos mais vendidos do ano por filial
C -> Os 3 produtos em que o cliente gastou mais dinheiro

q: fechar aplicacao

```

Menu Inicial

Queries

Este módulo contém todas as respostas a todas as queries e apresenta os resultados para o utilizador, definimos então a seguinte API para que fosse possível executar as queries no nosso *Navegador*.

```
int querieUmP (Cat_Prods catp);
int querieUmC (Cat_Cli catc);
int querieUmV (Faturacao fatp, Filial f, Cat_Prods catp, Cat_Cli catc);
int querieDois (Cat_Prods catp, char l, int x, int y);
int querieTres (Faturacao fatp, Filial f, Produto p, int mes);
int querieCinco (Filial f, Cat_Cli catc, int x, int y);
int querieQuatro (Faturacao fatp, Cat_Prods catp, int x, int y, int fil);
int querieSeis (Filial f, Faturacao fatp, Cat_Cli catc);
int querieSete (Filial f, Cliente c);
int querieOito (Faturacao fatp, int x, int y);
int querieNove (Filial f, Produto p, int filial, char tipo, int x, int y);
int querieDez (Filial f, Cliente c, int mes);
int querieOnze (Filial f, int filial, int n);
int querieDoze (Filial f, Cliente c);
```



API

→ Query 1:

- Implementamos funções de criação e validação, onde a função de criação aloca memória para a estrutura da informação extraída do ficheiro e a função de validação valida essa informação.

→ Query 2:

- Consoante a primeira, localizamos facilmente a GHashTable corresponde aquelas iniciais. Em seguida apenas precisamos de percorrer o array que dá informação da segunda letra e imprimir os elementos da GHashTable presente em cada uma dessas posições.

```
# de produtos comecados por A: 6578
Pagina: 1
AA1458 AA1603 AA1811 AA1201 AA1693 AA1777 AA1626 AA1079 AA1118 AA1308
AA1945 AA1649 AA1353 AA1996 AA1247 AA1627 AA1968 AA1292 AA1482 AA1711
AA1389 AA1270 AA1415 AA1203 AA1399 AA1630 AA1779 AA1969 AA1293 AA1226
AA1416 AA1081 AA1947 AA1165 AA1332 AA1545 AA1567 AA1629 AA1143 AA1333
AA1674 AA1713 AA1882 AA1121 AA1311 AA1501 AA1993 AA1736 AA1607 AA1144
AA1038 AA1650 AA1865 AA1083 AA1672 AA1295 AA1206 AA1357 AA1888 AA1927
AA1145 AA1296 AA1821 AA1229 AA1084 AA1274 AA1017 AA1380 AA1543 AA1464
AA1252 AA1486 AA1336 AA1503 AA1673 AA1985 AA1124 AA1169 AA1381 AA1845
AA1063 AA1504 AA1298 AA1784 AA1633 AA1041 AA1717 AA1125 AA1441 AA1209
AA1170 AA1064 AA1231 AA1632 AA1549 AA1824 AA1975 AA1232 AA1528 AA1763

p -> Proxima
a -> Anterior
b -> Voltar
CPU Time: 0.008504
```

→ Query 3:

- Conforme o mês e o tipo de venda recebido como argumento, localiza a GTree que contém toda a informação das vendas daquele produto naquele mês e daquele tipo. A partir daqui bastou utilizar funções da glib que procuram pela chave que, neste caso, é o código do produto e imprime o tamanho do array com as vendas bem como a faturação total com as vendas desse produto, nesse mês.

```
Produto: PE1822
Resultados Globais:
> Modo: Normal Mes: 3 Produto: PE1822 Vendas: 2 Faturado: 167925.81 €
> Modo: Promocao Mes: 3 Produto: PE1822 Vendas: 1 Faturado: 124674.72 €
CPU Time: 0.021625
```

→ Query 4:

- Antes de qualquer execução, é verificado se o array de não comprados presente na estrutura da Faturacao está vazio, se não estiver, imprime os resultados já calculados anteriormente. Se estiver vazio, vai procurar pelos produtos não comprados, sendo assim, com funções de travessia da glib, percorremos o catálogo de produtos e verificamos o número de vendas para esse produto.

```
# de produtos nao vendidos: 928
Pagina: 1
AA1143 AA1022 AA1236 AB1643 AB1432 AC1874 AC1285 AC1365 AD1865 AD1283
AE1280 AE1156 AF1094 AF1620 AG1065 AG1028 AI1782 AI1080 AI1123 AJ1051
AJ1262 AK1433 AL1932 AL1144 AL1344 AM1192 AN1791 AN1693 AO1576 AO1629
AP1182 AR1875 AS1139 AT1875 AV1603 AV1883 AV1927 AV1529 AW1504 AW1880
AX1657 AX1352 AZ1243 BA1039 BA1206 BA1974 BB1352 BC1307 BC1416 BF1787
BG1375 BH1212 BI1893 BJ1150 BL1746 BL1936 BM1938 BN1180 BP1868 BQ1191
BR1216 BS1760 BT1618 BU1766 BU1669 BX1879 BX1583 BY1837 CA1352 CB1532
CB1681 CC1817 CC1923 CE1926 CE1238 CE1505 CE1884 CF1527 CF1546 CF1508
CF1903 CG1616 CH1158 CH1436 CK1779 CL1149 CL1454 CL1761 CM1435 CN1464
CO1543 CP1445 CP1026 CQ1735 CR1134 CR1500 CR1273 CS1697 CS1579 CT1260

p -> Proxima
a -> Anterior
b -> Voltar
CPU Time: 0.014784
```

→ Query 5:

- Recebendo o catálogo de clientes, percorre todas as árvores com funções da glib e verifica se cada cliente vendeu em todas as filiais. Se vendeu então escreve-o no array presente na estrutura da Filial. O que facilita caso esta query seja usada várias vezes durante a execução do programa.

```
# de clientes compraram todas as filiais: 16384
Pagina: 1
A3339 A1349 A2129 A1295 A4652 A4496 A3520 A1428 A4090 A4222
A4866 A4575 A2639 A3442 A2741 A4091 A2077 A1531 A3013 A2718
A4146 A2820 A1454 A2078 A3014 A4224 A2389 A3445 A4093 A1377
A1533 A4734 A2822 A4304 A3470 A1456 A1534 A1612 A4579 A2589
A2745 A4095 A3017 A2823 A4149 A2770 A1536 A3525 A2184 A4815
A2903 A4253 A4331 A3958 A2748 A1992 A1538 A1616 A4098 A3959
A4739 A4841 A2187 A3984 A4378 A3476 A1642 A2697 A4257 A4281
A3125 A4688 A4790 A2932 A4259 A3205 A1801 A3127 A2856 A1998
A1724 A1240 A3989 A2935 A2779 A2881 A4417 A3207 A4847 A4848
A2858 A2802 A1726 A1804 A1594 A4012 A1242 A4443 A3209 A4521

p -> Proxima
a -> Anterior
b -> Voltar
CPU Time: 0.014021
```

→ Query 6:

- Esta query foi das mais simples de implementar porque apenas usamos duas funções, uma já usada na query 4, e outra que conta o número de clientes que não compraram através de uma travessia ao catálogo de clientes e Filial através de funções da glib.

```
> Numero de clientes que nao compraram: 0
> Numero de produtos nao comprados: 928
CPU Time: 0.418833
```

→ Query 7:

- Aqui é simples, basta procurar o cliente recebido em cada mês e em cada filial, percorrer o GPtArray e ir sempre somando o número de unidades vendidas. Imprimindo depois todas as informações de forma organizada.

```
Cliente: A3439
Janeiro: Filial 1-> 0 Filial 2-> 416 Filial 3-> 0
Fevereiro: Filial 1-> 194 Filial 2-> 122 Filial 3-> 330
Marco: Filial 1-> 0 Filial 2-> 264 Filial 3-> 0
Abril: Filial 1-> 178 Filial 2-> 0 Filial 3-> 211
Maio: Filial 1-> 141 Filial 2-> 299 Filial 3-> 62
Junho: Filial 1-> 23 Filial 2-> 184 Filial 3-> 0
Julho: Filial 1-> 462 Filial 2-> 402 Filial 3-> 414
Agosto: Filial 1-> 14 Filial 2-> 508 Filial 3-> 29
Setembro: Filial 1-> 0 Filial 2-> 265 Filial 3-> 0
Outubro: Filial 1-> 163 Filial 2-> 23 Filial 3-> 0
Novembro: Filial 1-> 140 Filial 2-> 88 Filial 3-> 0
Dezembro: Filial 1-> 302 Filial 2-> 563 Filial 3-> 0
CPU Time: 0.014361
```

→ Query 8:

- Percorre as arvores dos dois tipos naqueles meses recebidos como argumento com funções da glib e, para o número de vendas, calcula apenas o tamanho do array, para a faturação, percorre o array com as vendas e multiplica o preço unitário com as unidades vendidas.

```
> Numero de vendas entre [3...7]: 371217
> Total faturado entre [3..7]: 18626902016.00 €
CPU Time: 0.263647
```

→ Query 9:

- Usamos uma estrutura auxiliar, que tem um GArray e os argumentos da função de modo a ser possível fazer a travessia das GTree com as funções implementadas na glib. Assim, à medida que percorremos as GTree verificamos se o produto e o tipo de compra correspondem ao pedido pelo utilizador, se sim, guardamos no array.

```
# de clientes compraram produto PE1822 na filial 3 do tipo P: 2
Pagina: 1
V4505 A4588

p -> Proxima
a -> Anterior
b -> Voltar
CPU Time: 0.006047
```

→ Query 10:

- Foi criado um GPtrArray com uma estrutura que continha o código de produto e as unidades vendidas por este. Assim, para cada filial, é procurado o array correspondente às vendas do cliente passado como argumento. Para cada elemento do array, verificamos se o produto já foi adicionado ao GPtrArray auxiliar, se sim, apenas soma-se as unidades, se não, adicionamos o produto. No final, foi necessário criar uma função de comparação para que a glib ordenasse o array por ordem decrescente consoante as unidades vendidas. No final, simplificamos o GPtrArray para um GArray contendo só o código de produto e assim, poder ser imediatamente utilizado para imprimir no ecrã.

```
Produtos comprados pelo cliente A3439 no mes 12 por ordem decrescente de quantidade

1°-> SD1407 ----- 199
2°-> YD1491 ----- 185
3°-> JA1969 ----- 181
4°-> XR1993 ----- 179
5°-> TU1368 ----- 121
CPU Time: 0.012906
```

→ Query 11:

- Esta query é muito complexa. Inicialmente é iniciada uma GTree que contém em cada nodo uma struct auxiliar com código de produto, unidades vendidas e um GArray com todos os clientes que compraram aquele produto. Durante a travessia da Filial, o cliente só é inserido no array se não pertencer ao mesmo. Assim, no final, tal como na query anterior, é dado sort com funções de glib segundo as unidades vendidas e mais tarde passado para um GArray pronto a ser utilizado para imprimir no ecrã os resultados.

```
Produtos mais vendidos globais:

1°-> WX1593 --- unidades: 2071 ---- clientes: 15
2°-> WY1658 --- unidades: 2057 ---- clientes: 14
3°-> WS1798 --- unidades: 2034 ---- clientes: 15
4°-> XU1621 --- unidades: 2015 ---- clientes: 18
5°-> LA1356 --- unidades: 1958 ---- clientes: 15
6°-> WS1261 --- unidades: 1939 ---- clientes: 17
7°-> XX1529 --- unidades: 1935 ---- clientes: 16
8°-> HI1440 --- unidades: 1932 ---- clientes: 16
9°-> UI1645 --- unidades: 1931 ---- clientes: 15
10°-> JS1239 --- unidades: 1907 ---- clientes: 14
CPU Time: 2.563251
```

→ Query 12:

- Esta query foi pensada e estruturada estrategicamente da mesma maneira que a query 10, é criada uma estrutura auxiliar e é percorrida a informação. A maior diferença é que neste caso, guardamos o valor total gasto e não as unidades vendidas. No final, o GPtrArray é ordenado consoante o total gasto e é passado para um GArray pronto, novamente, a ser imediatamente utilizado para imprimir os resultados.

```
O cliente A3439 gastou mais dinheiro nestes 3 produtos

1°-> BS1662 ----- 153739.97 €
2°-> JE1700 ----- 153057.17 €
3°-> EC1706 ----- 141926.80 €
CPU Time: 0.016886
```

Profiling

Nos últimos dias do tempo limite dado pela equipa docente, conseguimos fazer um pouco de Profiling, no sentido de melhorar a eficiência e os resultados do nosso código, visto que reparamos estar a demorar demasiado tempo para carregar os ficheiros. Sendo assim conseguimos verificar alguns problemas no nosso código e corrigido alguns, apesar de termos noção que pelos resultados, muita coisa poderia ser melhorada. Alteramos os nossos catálogos de clientes e produtos para GHashTable devido ao tempo que demorava a validar as vendas e tivemos resultados significativos.

| Each sample counts as 0.01 seconds. | | | | | | |
|-------------------------------------|------------|---------|---------|---------|---------|------------------|
| % | cumulative | self | self | total | | |
| time | seconds | seconds | calls | ms/call | ms/call | name |
| 25.02 | 0.04 | 0.04 | 1000000 | 0.00 | 0.00 | validaVenda |
| 18.76 | 0.07 | 0.03 | 1000000 | 0.00 | 0.00 | criaVenda |
| 12.51 | 0.09 | 0.02 | 1171008 | 0.00 | 0.00 | criaProduto |
| 12.51 | 0.11 | 0.02 | 891108 | 0.00 | 0.00 | insereFilV |
| 6.25 | 0.12 | 0.01 | 2583165 | 0.00 | 0.00 | getProdVenda |
| 6.25 | 0.13 | 0.01 | 1782216 | 0.00 | 0.00 | getMes |
| 6.25 | 0.14 | 0.01 | 1782216 | 0.00 | 0.00 | getTipoVenda |
| 6.25 | 0.15 | 0.01 | 891108 | 0.00 | 0.00 | insereVenda |
| 3.13 | 0.16 | 0.01 | 901565 | 0.00 | 0.00 | existeProd |
| 3.13 | 0.16 | 0.01 | 171008 | 0.00 | 0.00 | insereProd |
| 0.00 | 0.16 | 0.00 | 1782216 | 0.00 | 0.00 | getFilial |
| 0.00 | 0.16 | 0.00 | 1782216 | 0.00 | 0.00 | getPrecoVenda |
| 0.00 | 0.16 | 0.00 | 1782216 | 0.00 | 0.00 | getUniVenda |
| 0.00 | 0.16 | 0.00 | 1350792 | 0.00 | 0.00 | getCliVenda |
| 0.00 | 0.16 | 0.00 | 1243581 | 0.00 | 0.00 | getProduto |
| 0.00 | 0.16 | 0.00 | 1072573 | 0.00 | 0.00 | primeiraLetra |
| 0.00 | 0.16 | 0.00 | 1072573 | 0.00 | 0.00 | segundaLetra |
| 0.00 | 0.16 | 0.00 | 1032768 | 0.00 | 0.00 | getClient |
| 0.00 | 0.16 | 0.00 | 1016384 | 0.00 | 0.00 | criaCliente |
| 0.00 | 0.16 | 0.00 | 1016384 | 0.00 | 0.00 | primeiraLetraCli |
| 0.00 | 0.16 | 0.00 | 1000000 | 0.00 | 0.00 | existeCli |
| 0.00 | 0.16 | 0.00 | 171008 | 0.00 | 0.00 | validaProduto |
| 0.00 | 0.16 | 0.00 | 16384 | 0.00 | 0.00 | insereCli |
| 0.00 | 0.16 | 0.00 | 16384 | 0.00 | 0.00 | validaCliente |
| 0.00 | 0.16 | 0.00 | 1 | 0.00 | 0.00 | querieUmC |
| 0.00 | 0.16 | 0.00 | 1 | 0.00 | 160.10 | querieUmMenu |
| 0.00 | 0.16 | 0.00 | 1 | 0.00 | 7.93 | querieUmP |
| 0.00 | 0.16 | 0.00 | 1 | 0.00 | 152.18 | querieUmV |

Quando o CatProds e o CatCli eram compostos por GTree os resultados eram estes

| Each sample counts as 0.01 seconds. | | | | | | |
|-------------------------------------|------------|---------|---------|---------|---------|------------------|
| % | cumulative | self | calls | self | total | |
| time | seconds | seconds | | ms/call | ms/call | name |
| 50.03 | 0.27 | 0.27 | | | | prod_n |
| 12.97 | 0.34 | 0.07 | 721028 | 0.00 | 0.00 | elemArr |
| 9.27 | 0.39 | 0.05 | | | | cmp_n_h |
| 5.56 | 0.42 | 0.03 | 891108 | 0.00 | 0.00 | insereVenda |
| 3.71 | 0.44 | 0.02 | 1000000 | 0.00 | 0.00 | criaVenda |
| 3.71 | 0.46 | 0.02 | 1000000 | 0.00 | 0.00 | validaVenda |
| 3.71 | 0.48 | 0.02 | 891108 | 0.00 | 0.00 | insereFilV |
| 2.78 | 0.50 | 0.02 | 1016384 | 0.00 | 0.00 | criaCliente |
| 2.78 | 0.51 | 0.02 | 171008 | 0.00 | 0.00 | validaProduto |
| 1.85 | 0.52 | 0.01 | 1782216 | 0.00 | 0.00 | getTipoVenda |
| 1.85 | 0.53 | 0.01 | 1171008 | 0.00 | 0.00 | criaProduto |
| 1.85 | 0.54 | 0.01 | 901565 | 0.00 | 0.00 | existeProd |
| 0.00 | 0.54 | 0.00 | 2583165 | 0.00 | 0.00 | getProdVenda |
| 0.00 | 0.54 | 0.00 | 1782216 | 0.00 | 0.00 | getFilial |
| 0.00 | 0.54 | 0.00 | 1782216 | 0.00 | 0.00 | getMes |
| 0.00 | 0.54 | 0.00 | 1782216 | 0.00 | 0.00 | getPrecoVenda |
| 0.00 | 0.54 | 0.00 | 1782216 | 0.00 | 0.00 | getUniVenda |
| 0.00 | 0.54 | 0.00 | 1350792 | 0.00 | 0.00 | getCliVenda |
| 0.00 | 0.54 | 0.00 | 1243581 | 0.00 | 0.00 | getProduto |
| 0.00 | 0.54 | 0.00 | 1072573 | 0.00 | 0.00 | primeiraLetra |
| 0.00 | 0.54 | 0.00 | 1072573 | 0.00 | 0.00 | segundaLetra |
| 0.00 | 0.54 | 0.00 | 1032768 | 0.00 | 0.00 | getClient |
| 0.00 | 0.54 | 0.00 | 1016384 | 0.00 | 0.00 | criaCliente |
| 0.00 | 0.54 | 0.00 | 1000000 | 0.00 | 0.00 | existeCli |
| 0.00 | 0.54 | 0.00 | 171008 | 0.00 | 0.00 | validaProduto |
| 0.00 | 0.54 | 0.00 | 16384 | 0.00 | 0.00 | insereCli |
| 0.00 | 0.54 | 0.00 | 16384 | 0.00 | 0.00 | validaCliente |
| 0.00 | 0.54 | 0.00 | 1 | 0.00 | 0.00 | querieUmC |
| 0.00 | 0.54 | 0.00 | 26 | 0.00 | 0.00 | getProdutosLetra |
| 0.00 | 0.54 | 0.00 | 1 | 0.00 | 0.00 | prodNcomp |
| 0.00 | 0.54 | 0.00 | 1 | 0.00 | 0.00 | produtosPorLetra |
| 0.00 | 0.54 | 0.00 | 1 | 0.00 | 0.00 | querieDoisMenu |
| 0.00 | 0.54 | 0.00 | 1 | 0.00 | 0.00 | querieOnze |
| 0.00 | 0.54 | 0.00 | 1 | 0.00 | 0.00 | querieOnzeMenu |
| 0.00 | 0.54 | 0.00 | 1 | 0.00 | 0.24 | querieUmC |
| 0.00 | 0.54 | 0.00 | 1 | 0.00 | 150.09 | querieUmMenu |
| 0.00 | 0.54 | 0.00 | 1 | 0.00 | 16.47 | querieUmP |
| 0.00 | 0.54 | 0.00 | 1 | 0.00 | 133.38 | querieUmV |

Demasiado tempo passado numa função responsável pela query 11, seria um caso a estudar com tempo

Conclusão

Concluído o trabalho, entendemos que a solução apresentada é eficiente e bem estruturada. Foram efetuadas várias medições de tempo, para além do *profiling*, que comprovam a boa performance das abordagens seguidas nas várias queries. A nível de memória, houve o cuidado de dar *free* das estruturas usadas para evitar redundâncias e alocações de memória desnecessárias. Houve certamente uma aprendizagem notória relativamente às estruturas implementadas da biblioteca glib, e tentou fazer-se o uso máximo dessas estruturas, tirando assim partido da estabilidade e a fiabilidade das mesmas. Apesar de muito código reescrito e melhorado, haveria certamente espaço para melhorar, como o caso da query 11.

No geral faz-se uma avaliação muito positiva das diversas métricas consideradas e entende-se que a abordagem ao problema foi concisa e direta.