Conteúdo

Capítulo 1

Introdução

Este trabalho, desenvolvido na linguagem de programação C, resume-se ao tratamento de dados de uma aplicação de viagens.

Uma vez que este projeto operava com grandes porções de dados e para conseguirmos o armazenamento eficiente dos mesmos, recorremos a conceitos de programação mais complexos como estruturas de dados, nomeadamente Hash Tables que pertencem à biblioteca GLib da GNOME.

Inicialmente o nosso maior obstáculo consistiu em armazenar os dados de forma correta pois a não utilização de algumas estruturas levou a uma alta taxa de memory leaks a responder às queries prejudicando assim o desempenho do programa. Ultrapassado este contratempo e definida a lógica do nosso programa seguimos a arquitetura apresentada, tendo sempre em conta o encapsulamento.

Capítulo 2

Módulos e estruturas de dados

2.1 - Listas

As listas, ou catálogos, definidos nestes três módulos são bastante semelhantes porém necessitaram de ser divididas:

2.1.1 - Lista de condutores

Parsing do catálogo dos condutores, organizados numa estrutura, estes valores são apenas acessáveis pelas funções também definidas neste módulo

2.1.2 - Lista de utilizadores

Neste módulo são armazenados os condutores no seu catálogo durante o parsing dos mesmos.

2.1.3 - Lista de viagens

Apesar da lógica deste módulo ser semelhante aos anteriores, envolve uma complexidade acrescida, pelo que foi dividido num módulo complementar, *RidesA*, pois para além das funções de acesso ao catálogo, a resolução das queries é mais dependente dos outros dois catálogos.

Cade um destes módulos possui funções próprias que lhes permitem aceder aos dados das tabelas Hash, os “getters”.

2.2 - Estatísticas

Esta parte do programa é responsável por orquestrar as suas funções de forma a responder às queries. É um módulo totalmente dependente dos catálogos porém o encapsulamento foi levado em conta, também por uma questão de eficiência, reduzimos ao máximo os inputs e outputs de cada função.

2.3 - Resultados

Este módulo participa na entrega dos resultados obtidos no módulo anterior à função principal, explicado com mais detalhes posteriormente

2.4 - Auxiliares

O módulo *sameStats*, participa em algumas operações complementares como parsing da data em queries que o exijam, assim como, principalmente, na conversão de tipo de dados. Estas funções são transversais a todos os módulos.

Capítulo 3

Estrutura do projeto

Nesta primeira fase do projeto, organizamos o mesmo em 3 grupos:

- Organização em catálogos: o parsing de cada lista será feito pelo seu respetivo módulo, sendo ele *list_driver*, *list_user* ou *list_rides*, este último usa o módulo auxiliar RidesA. Estes dois últimos módulos mencionados têm o mesmo propósito, gerar o catálogo das viagens e as estatísticas alusivas às mesmas, no entanto por uma questão de distribuição de código foram divididos.

- Tratamento de dados: o módulo stats organiza os catálogos numa struct própria, calcula estatísticas úteis e procede corretamente à resolução do problema proposto em cada querie, colocando o resultado atribuído numa estrutura denominada *row* que por sua vez será colocada noutra estrutura que consiste num conjunto de *row's*, construindo assim cada output.

- Resultados: este módulo, coordenado com a função principal, é responsável por orquestrar as funções que respondem às respetivas queries, apresentando o resultado no modo de funcionamento indicado, atualmente apenas em batch.

Este projeto foi assim modulado para uma melhor compreensão por parte do programador. Epilogando, de um modo geral, a função principal aciona a criação dos catálogos e solicita os resultados pedidos no input, estes por sua vez são determinados no módulo stats em conjunto com os catálogos gerados e por fim apresentados ao utilizador em modo batch. Todo este processo leva em conta o encapsulamento de dados.

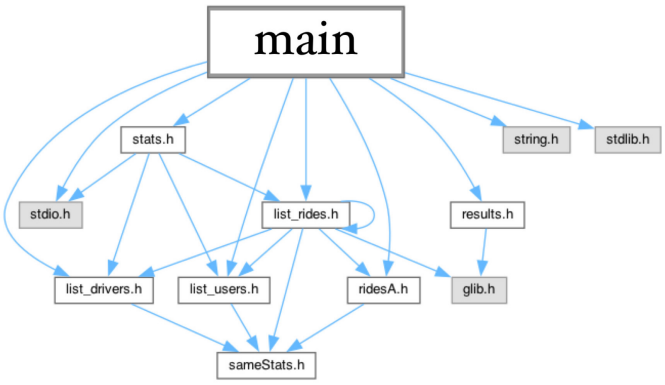

Figura 1 - representação gráfica estrutural das interações entre módulos (gerado pelo Doxygen)

Capítulo 4

Queries

- **Querie 1**

Para responder a esta querie, o programa procede ao calculo de algumas estatísticas, como avaliação média, número de viagens e total gasto.

Comando: 1 *<ID>*

Output: nome;genero;idade;avaliacao_media;numero_viagens;total_gasto

- **Querie 2**

Devolve uma lista com os N condutores com maior avaliação média.

Comando: 2 *<N>*

Output (lista): **i**d;nome;avaliacao_media

- **Querie 4**

O programa calcula o preço médio por viagem numa determinada cidade.

Comando: 4 <city>

Output (lista): **i**d;nome;avaliacao_media

Capítulo 5

Complexidade das estruturas

De modo a minimizar o impacto do programa na memória, recorremos a estruturas de dados para armazenar a quantidade avultada dos mesmos. Dependendo do cenário, adotamos tabelas Hash e estruturas de estruturas de dados, denominadas árvores binárias.

Cada catálogo define uma estrutura com o tipo de dados com que está a trabalhar, assim como os tipos de dados obtidos nos ficheiros csv. Este método permite que sejam devolvidos às restantes funções os dados do utilizador, condutor ou da viagem de forma mais rápida.

No calculo das estatísticas, definimos uma estrutura que contém objetos do tipo dos três catálogos existentes, estes serão preenchidos com os dados ou cálculos pedidos em cada querie. Estas estatísticas são armazenadas numa outra estrutura, *Results,* que contém várias linhas (rows), resultando assim numa estrutura com os outputs obtidos, como mencionado no capítulo anterior.

Também é de salientar que estatísticas mais especificas e que envolvem uma interdependência de catálogos necessitaram de estruturas próprias, por exemplo, para calcular o preço de uma viagem é necessário acessar todos os catálogos para obtermos a distância da viagem, a tarifa do condutor e que utilizador fez essa viagem, neste caso e em semelhantes, recorremos à estrutura *ud_stats.* Similarmente em estatísticas referentes a cidades recorremos a estruturas próprias facilitando assim a organização do programa.

Capítulo 6

Conclusão

Nesta primeira fase do projeto, como já explicado nos capítulos anteriores, o programa possui estruturas robustas e, até agora, eficientes. Este resultado apenas foi conseguido após algumas tentativas falhadas de organização de dados e um uso imenso da memória numa fase rudimentar do projeto.

Apesar de não termos as queries todas resolvidas, e sem nenhum benchmark oficial, apenas comparando com versões anteriores do nosso projeto e pelo monitor de atividade do sistema operativo, verificamos que o programa tem um desempenho bem aceitável para a quantidade de dados da primeira fase.