Relatório

Trabalho Prático - Fase 1

Grupo 44

Hugo Arantes Dias (a100758)

Ricardo Miguel Queirós de Jesus (a100066)

Rui Pedro Fernandes Madeira Pinto (a100659)

Departamento de Informática

Licenciatura em Engenharia Informática

2º ano | 1º Semestre

Laboratórios de Informática III

ÍNDICE

1 Introdução

O presente relatório tem como objetivo, descrever em detalhe, o trabalho desenvolvido no âmbito da unidade curricular de Laboratórios de Informática III, lecionada no curso de Licenciatura em Engenharia Informática na Universidade do Minho, no 1º semestre do ano letivo de 2022/2023.

O grupo 44 é composto pelos elementos:

- Hugo Arantes Dias a100758;

- Ricardo Miguel Queirós de Jesus a100066;

- Rui Pedro Fernandes Madeira Pinto a100659;

2 Objetivos

Resumidamente, o trabalho prático tem como objetivos, efetuar a leitura dos 3 ficheiros de entrada (users.csv, drivers.csv e rides.csv) e tratar dos dados dos mesmos de modo a executar as respetivas queries com os argumentos indicados.

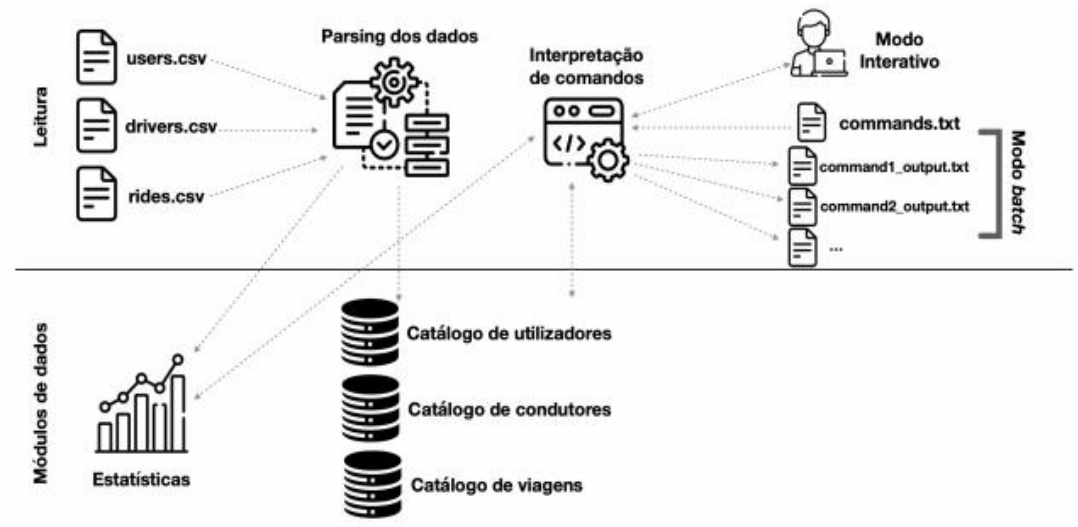

Figura 1: Arquitetura de referência para a aplicação a desenvolver

3 Modularização/Encapsulamento

A pasta **trabalho-pratico/** submetida no *GitHub* apresenta a seguinte estrutura:

- **trabalho-pratico/**
  - Makefile
  - hash.c
  - hash.h
  - main.c
  - loadGeral.c
  - loadGeral.h
  - parserDrivers.c
  - parserDrivers.h
  - parserRides.c
  - parserRides.h
  - parserUsers.c
  - parserUsers.h
  - querie2.c
  - querie2.h
  - querie3.c
  - querie3.h
  - querie4.c
  - querie4.h
  - queries.c
  - queries.h
  - queries.txt
  - structs.c
  - structs.h
  - date.c
  - date.h
  - avaliacao_driver.c
  - avaliacao_driver.h
  - **Resultados/**

O trabalho é composto por vários módulos, tendo sido os que se seguem, criados consoante as necessidades que se foram detetando.

- **main.c -** main do programa;

- **hash.c -** contém todas as funções relativas às hash tables;
- **parserDrivers.c -** contém as funções para parsing do ficheiro drivers.csv;
- **parserRides.c** - contém as funções para parsing do ficheiro rides.csv**;**
- **parserUsers.c** - contém as funções para parsing do ficheiro users.csv;
- **querie2.c** - contém todas as funções usadas para responder à query 2;
- **querie3.c** – contém todas as funções para responder à query 3;
- **querie4.c** – contém todas as funções para responder à query 4;
- **queries.c** – contém todas as funções para ler e escrever em ficheiros.txt**;**
- **structs.h** – contém todas as structs usadas no projeto;
- **date.c –** contém as funções para trabalhar com as datas;
- **loadGeral :** contém as funções para dar init e load à struct geral;
- **avaliacao_driver:** contém as funções para trabalhar com a struct aval_driv;

Os módulos criados por nós complementam-se, cada um com objetivos específicos como por exemplo, aceder a estruturas de dados ou responder a queries, de forma a facilitar a reutilização do código.

- Encapsulamento

A nossa estratégia de encapsulamento passa por definir cada módulo com funções específicas e que não podem ser executadas por outros módulos diretamente, mas sim através de gets e sets que vão ao módulo indicado executar a função necessária ou obter a informação desejada.

Em termos práticos, pretendemos estruturar o resto do trabalho da mesma forma que fizemos para resolver a query 2.

4 Estruturas de Dados

O grupo decidiu criar as seguintes *structs* com o objetivo de responder a cada uma das *queries* da forma mais eficiente possível:

```
typedef struct driver
{
        char *id;
        char *nome;
        char *birthday;
        char *gender;
        char *carclass;
        char *city;
        char *account_creation;
        char *account_status;
        int viagens;
        int avaliacao;
        int last_ride;
} * Driver;
```

A estrutura de dados **Driver** contém o ID, nome, data de nascimento, género, classe de veículo, cidade, data de criação da conta e status da conta, contém também o número total de viagens do driver em questão e a soma das avaliações que o driver teve em cada viagem, estas duas últimas variáveis servem para nos ajudar a fazer a avaliação média utilizada nas queries posteriormente, ainda acrescentamos a last_ride para saber quando efetuou a última viagem do Driver

```
typedef struct ride
{
        char *id;
        char *data;
        char *idDriver;
        char *username;
        char *city;
        char *distance;
        char *score_user;
        char *score_driver;
        char *tip;
        char *comment;
} * Ride;
```

A estrutura de dados **Ride** contém o id, a data, a cidade, distância e a avaliação do *user* e do *driver* da respetiva viagem. Esta estrutura contém também o id do condutor, o nome de utilizador do *user*, a gorjeta e o comentário deixado pelo mesmo.

```
typedef struct user
{
        char *username;
        char *name;
        char *gender;
        char *birth_date;
        char *account_creation;
        char *pay_method;
        char *account_status;
        int distance;
        int last_ride;
} * User;
```

A estrutura de dados **User** contém o nome de utilizador, nome, género, data de nascimento, data de criação de conta, método de pagamento, estado da conta , distância viajada pelo user e o quando efetuou a última viagem.

```
typedef struct city
{
        char *cityname;
        double preco;
        int viagens;
} * City;
```

A estrutura de dados **City** contém o nome da cidade, o preço total de todas as viagens da cidade em questão, e o número de viagens efetuadas na mesma. Estes dois últimos dados são armazenados para que consigamos fazer o preço médio de cada viagem na cidade de forma mais fácil nas queries em que tal informação é pedida.

```
typedef struct geral
{
        GHashTable *hashDriv;
        GHashTable *hashUser;
        GHashTable *hashRides;
        GHashTable *hashCity;
} * Geral;
```

A estrutura de dados **Geral** serve para armazenar todas as hash tables das quais necessitamos para armazenar a informação necessária para responder aos exercícios. Temos uma HashTable para Drivers, Users, Rides e Cidades .

```
typedef struct aval_driver
{
        char *id;
        char *name;
        double avaliacao_med;
        int last_ride_days;
} * Aval_Driver;
```

A estrutura **Aval_Driver** é uma struc auxiliar que serve para a facilitar a introdução dos dados na querie 2, em que temos, para um driver, o seu id, nome, avaliação média e quando efetuou a sua última viagem (em dias).

```
typedef struct dist_user
{
        char *username;
        char *name;
        int distance;
        int last_ride_days;
} * Dist_User;
```

A estrutura de dados **Dist_User** é uma struct auxiliar que serve para facilitar a ordenação da querie 3, em que temos, para um User, o seu username, nome, distancia percorrida e quando efetuou a sua última viagem (em dias).

5 Queries

Neste trabalho prático, estão presentes 9 queries às quais temos de executar, contudo, na fase 1 são necessárias, no mínimo, apenas 3 queries funcionais. O nosso grupo optou por fazer as queries 2, 3 e 4.

Query 2:

A query 2 recebe como argumento um número N, e lista os N condutores com maior avaliação média.

Para responder à query 2 temos uma função que conta todas as viagens do driver dado e depois dá o valor à variável viagens da struct Driver. Outra que calcula a soma de todas as avaliações de todas as viagens efetuadas pelo driver abordado. Uma que dado um driver x calcula a sua avaliação média usando as variáveis ‘viagens’ e ‘avaliacao’. Uma função top_N_Drivers_Aval que usa uma auxiliar compare_drivers_aval que serve para ir comparando os drivers ao longo da lista tendo em conta os casos de desempate. Usamos a avaliação média para ordenar os Drivers, em caso de empate, colocamos primeiro quem teve a viagem mais recente, e se ainda assim houver outro empate desempatamos pelo id do Driver, colocando o que tem o menor id a cima. Voltando à função top_N_Drivers_Aval, antes de usarmos a auxiliar, verificamos se o driver está ativo.

Query 3:

A query 3 recebe como argumento um número N, e lista os N utilizadores com maior distância viajada.

Para responder à query 3 temos uma função responsavel por ir somando todas as distancias viajadas pelo user em todas as suas viagens e ir incrementando a variável 'distance' na struct User. Uma função top_N_User_Dist que usa uma usa a função compare_users_dist para ordenar os User por distancia percorrida, em caso de empate, colocamos primeiro quem teve a viagem mais recente, e se ainda assim houver empate desempatamos pelo username colocando os mesmos por ordem alfabetica. Voltando à função top_N_User_Dist, antes de usarmos a auxiliar, verificamos se o user está ativo.

Query 4:

A query 4 recebe como argumento o nome de uma cidade e devolve o preço médio das viagens (sem considerar as gorjetas) na mesma.

Para responder à query 4 temos uma função que conta as viagens todas efetuadas na cidade dada e depois dá o valor à variável viagens da struct **City** e outra função que calcula a soma de todos os preços das viagens efetuadas na cidade abordada (obviamente tendo em conta os parâmetros que vão interferir na mesma, nomeadamente a *carclass* e a distância da viagem). Esta última vai incrementando a variável **preco** na struct **city**, de forma que esta fique com o preço total das viagens no fim. Temos ainda uma terceira função que dada uma cidade dá o preço médio por viagem da mesma. Finalmente, temos uma função que dá o output da querie 4 para o mesmo ser escrito no ficheiro, ainda nesta função efetuamos a criação do ficheiro **commandX_output.txt**. Sabemos que esta última não deveria estar neste local e vamos passar isto para outro ficheiro de forma a melhorar o encapsulamento na próxima fase do projeto.

6 Hash Tables

Antes de escolhermos esta estrutura de dados, surgiram outras ideias, nomeadamente a de optar por usar *arrays*, mas tornaria-se demasiado ineficiente, visto que para conseguirmos obter as informações necessárias para as queries teríamos de percorrer o array à procura da informação, o que seria bastante indeficiente, tendo em conta o tamanho das bases de dados que nos são fornecidas. Por estes motivos optamos pelas *Hash Tables*.

No caso da Hash Tables o processo fica mias efeciente. Ainda que a Hash Table possa demorar mais tempo a "montar" acaba por compensar pois podemos associar values a keys conhecidas facilitando, prosteriormente a procura na base de dados, visto que podemos ir diretamente à informação desejada.

7 Modo Batch

Para executar o modo batch escrevemos todas as funções no ficheiro 'queries.c'. Neste ficheiro temos, a função principal que percorre todos os comandos presentes no ficheiro 'queries.txt' (onde são passados os comandos para execução das queries). Para a leitura deste ficheiro e armazenamento dos dados usamos duas funções auxiliares. Uma delas guarda o número da query em questão outra guarda os argumentos dados à mesma. Após termos estas informações usamos um switch case para verificar o número da query a executar, e executamos a query pedida pelo comando em questão. A esta query vamos dar os argumentos que o comando dá e o número do comando a ser executado para facilitar depois a criação do ficheiro com o seu número. Para a criação destes ficheiros temos 3 funções. Uma delas serve para criar o ficheiro quando o output tem de ser uma string, outra delas serve para quando o output tem de ser em double e uma última usada quando no caso da query 4 a cidade não possui qualquer viagem.

8 Conclusão

No geral, o grupo autoavalia-se positivamente visto que concluímos grande parte do que nos foi proposto para entregar na primeira fase deste projeto.

A parte do projeto onde o nosso grupo encontrou mais dificuldades foi em realizar o mesmo mantendo um encapsulamento ideal.

Além dessa parte específica, também consideramos que foi difícil aprender a trabalhar com todas as informações da *glib* visto haver pouca informação/exemplos onde a mesma é usada. Além disso inicialmente também tivemos alguma dificuldade em encontrar a melhor maneira possível de fazer o parsing dos ficheiros para variáveis dentro das structs respetivas a cada tipo de dado.