
Universidade do Minho
Licenciatura em Engenharia Informática

Fase 1 e 2

# Laboratórios de informática III 2022/23

## Grupo 1:

Afonso Martins (A96452)
André Alves (a95033)
Gonçalo Freitas (A96136)



5 de fevereiro de 2023

# Conteúdo



# Lista de Figuras

# 1 Introdução

Ao longo deste relatório é abordado o desenvolvimento do trabalho, centrando a atenção na resolução das etapas propostas pelos docentes neste projeto, assim como nas estratégias e métodos utilizados. Após a primeira fase, foi-nos proposta uma abertura de horizontes no que seriam os nosso métodos estruturais para este desenvolvimento, apresentando assim o trabalho que está aqui hoje.

# 2 Desenvolvimento do trabalho

Nesta primeria fase do trabalho de Laboratórios de Informática III deparamos agora com a necessidade de criar uma arquitetura para uma API. Onde nos era aconselhado a separação em 'Leitura' e 'Módulos de Dados', como podemos observar na Figura 2.1 (Figura abaixo).

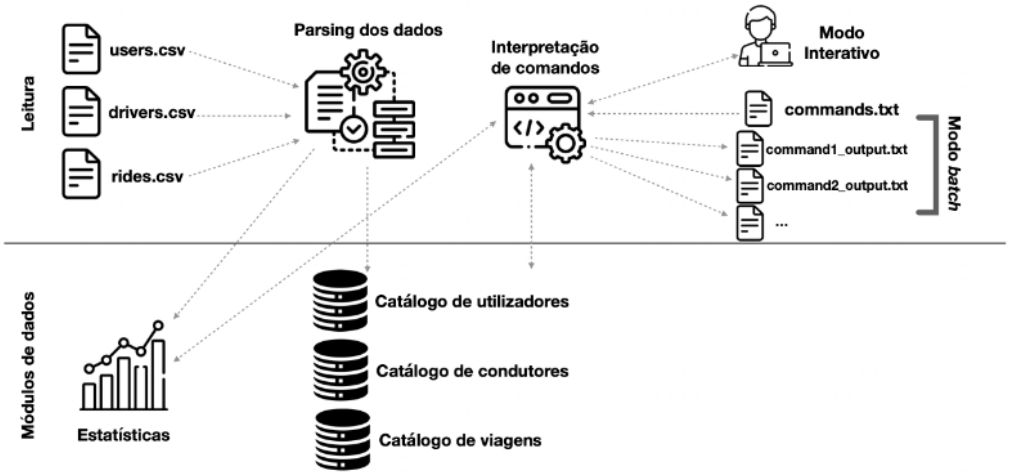


Figura 1: Arquitetura de referência

Após a leitura dos ficheiros '*.csv' é necessário guardar os dados importantes para a realização das queries. Após a leitura necessitamos de armazenar os dados, na qual situação optamos por utilizar vários tipos de hashtable, como explicado.Para alguns valores necessários às queries, iniciaremos a sua realização durante a execução da criação dos catálogos, tais que serão mencionados posteriormente, pois desta forma conseguimos otimizar o programa fazendo com que não seja necessária a travessia dos dados na sua totalidade múltiplas vezes. Finalizando assim com a interpretação do ficheiro de comandos que resulta no output de vários ficheiros consoante o número de queries solicitadas, consoante o input.

## 2.1 Hashtable vs Outros métodos

Após o teste de outras estruturas de dados como árvores binárias e listas ligadas, o tempo de inserção pós ordem teria um custo muito mais elevado do que numa hashtable. Como o número de input de comandos não é assim tão elevado, não compensa a criação de uma árvore binária devido à sua organização demorada.

## 2.2 Encapsulamento e Modularidade

O encapsulamento e a modularidade apenas nos permite aceder aos módulos, 'user.c', 'users.c', 'driver.c', 'drivers.c', 'ride.c', 'rides.c', 'hashtable.c', 'queries.c' e 'parser.c' através da sua API, promovendo o encapsulamento.

### 2.3 Catálogos

Optando assim pela utilização como armazenamento de dados de uma hashtable, criamos hashtables para os ficheiro dos 'users', 'drivers' e 'rides'. A criação destes por este método torna o seu tempo de execução linear.

Aqui, cada linha do ficheiro é diferenciada pelo seu respetivo id e é guardado na tabela consoante a função hash da key(id). Ficheiros em que as keys são diferentes e as hash tem o mesmo valor, é solucionado como se fosse uma espécie de lista ligada inserida na hash.

### 2.4 Validação

Para esta fase do trabalho foi necessário analisar o conteúdo dos vários campos(colunas) presentes em cada linha do ficheiro e verificar se este era válido e/ou o formato correspondia àquele exigido. Para isso criamos as funções que nos permitem testar a validade dos campos, onde com estas implementadas, na criação dos catálogos de cada um dos tipos se um campo de uma linha não estivesse correto essa linha não seria inserida no seu respetivo catálogo.

Todas as funções de validação, *is_valid_user*, *is_valid_driver* e *is_valid_ride*, têm o mesmo comportamento, cada uma recebe a linha que será a candidata a ser inserida no catálogo, que já se encontra no formato capaz de ser possível buscar cada um dos campos necessários a validar, e realiza a cada um dos campos a função de validação necessária consoante o enunciado nos requer realizar. Apenas se passar por todos estes testes a função irá retornar um inteiro apropriado à sua validação, no caso de ser válida retornará 1, o que na função de inserção desta linha no catálogo irá apenas a inserir se receber essa confirmação, não a colocando se receber um inteiro 0.

## 3 Especificações

Como requerido no relatório, iremos anunciar as especificações dos computadores utilizados no teste e desenvolvimente deste projeto.

### 3.1 Afonso Martins (1)

Processador: 1,1 GHz Intel Core i5 de núcleo quádruplo

Placa Gráfica: Intel Iris Plus Graphics 1536 MB

RAM: 8 GB

Discos: SSD 512GB

### 3.2 André Alves (2)

Processador: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz

Placa Gráfica: AMD Radeon Vega 8 Graphics

RAM: 8GB

Discos: SSD 512GB

### 3.3 Gonçalo Freitas (3)

Processador: Apple M2

Placa Gráfica: Intel Iris Plus Graphics 645

RAM: 8 GB

Discos: SSD 512GB

### 3.4 Ficheiro de Input

Seguidamente iremos partilhar os ficheiros de comandos que utilizamos para fazer testes e testar tempos nesta fase do trabalho.

1 PetrPacheco

1 000000002639

4 Setúbal

5 01/01/2021 01/01/2021

6 Setúbal 02/01/2021 08/01/2021

8 M 11

8 F 11

## 4 Queries

As queries são comandos executados através de uma constante, id da querie, e algumas por uma constante e certos argumentos, tais argumentos variando de query para query. Nestas queries existem parâmetros que são facilmente calculados na criação dos catálogos respetivos a cada ficheiro, logo como mencionado atrás estes valores serão logo criados juntamente com os catálogos globais, por exemplo as tabelas ht_user_ride e ht_driver_ride, que são usadas para a primeira query, são geradas juntamente com a criação das tabelas globais pois feito de outra forma seria necessária computação desnecesária.

### 4.1 Query 1

Nesta query nós consoante o id que recebermos, caso seja um número é um driver e caso seja uma string é um user, iremos retornar das seguintes formas respetivamente:

1. User: nome;genero;idade;avaliacao_media;numero_viagens;total_gasto
2. Driver: nome;genero;idade;avaliacao_media;numero_viagens;total_auferido

Os parâmetros acima serão, para além dos obviamente detetáveis, a avalicao media que o user dá ou que o driver recebe consoante todas as viagens que fizeram, o número de viagens que o user ou que o driver realizaram e o total que o user gastou ou que o driver recebeu por todas as suas viagens. Caso um user ou driver esteja ou inativo ou até que não exista nos nossos catálogos a query não irá retornar nada.

Para a realização desta query ao invés de percorrermos o catálogo necessário uma vez para cada parâmetro,sendo que isso ia ter um custo demasiado elevado, ao realizarmos o cálculo do parâmetro da avaliacao_media cálculamos juntamente o número de viagens que ele realizou e o total_gasto ou total_auferido, dependendo do id. Desta forma ao invés de percorrermos 3 vezes a tabela percorremos a percorremos uma vez.

Tempo de execução – N - Linear.

(1) => 0.000324s e 0.000138s

(2) => 0.002s e 0.003s

(3) => 0.000196s e 0.000063s

## 4.2 Query 2

Nesta query, consoante o $n$ que recebermos, devolvemos os $n$ condutores com maior avalição média. Em caso de empate, o resultado deverá ser ordenado de forma a que os condutores com a viagem mais recente surjam primeiro. Caso haja novo empate, é usado o id do condutor, por ordem crescente, para desempatar. Retorna os dados no seguinte formato:

1. id;nome;avaliacao_media

Os parâmetros acima serão, o id do *driver*, o nome do *driver* e a avaliação média que o driver recebe consoante todas as viagens que fez.

Para a realização desta query percorremos o catálogo dos drivers e, de cada driver, com o auxílio da função **get_avaliacao_media_driver**, obtíamos o valor da avaliação média do condutor em questão.

## 4.3 Query 3

Nesta query, após receber o $n$, devolvemos o *user* dos utilizadores com maior distância viajada. Em caso de empate, o resultado deverá ser ordenado de forma a que os utilizadores com a viagem mais recente surjam primeiro. Caso haja novo empate, é usado o username do utilizador, por ordem crescente, para desempatar. Retorna os dados no seguinte formato:

1. username;nome;distancia_total

Os parâmetros acima serão, o *username* dos utilizadores com maior distância viajada, o nome do *driver* e a distância total percorrida pelos utilizadores, após a soma da distâncias trilhadas.

Os parâmetros acima serão, o id do *driver*, o nome do *driver* e a avaliação média que o driver recebe consoante todas as viagens que fez.

Para a realização desta query percorremos o catálogo dos drivers e, de cada driver, com o auxílio da função **get_avaliacao_media_driver**, obtíamos o valor da avaliação média do condutor em questão.

## 4.4 Query 4

Esta query apenas recebe o dado *city* de possível passagem e devolvemos o *preco_medio* dessas viagens a passar pela cidade em questão, mas sem considerar possíveis gorjetas. Retorna os dados no seguinte formato:

1. preco_medio

O parâmnetro acima mencionado, o preço médio das viagens, sem considerar gorjetas, *preco_medio*, na cidade que for dada ao evocar a mesma função.

Para a realização desta query, estudamos todas as viagens realizadas numa determinada cidade, depois, consoante a classe de veículo, *car_class*, calculamos o preço das viagens, já que os carros podem variar entre *Basic*, *Green* e *Premium*. Depois, só aì, calculamos a média de tudo.

(1) => 0.599340s

(2) => 1.236s

(3) => 0.120677s

## 4.5 Query 5

Nesta query, recebemos dois parâmetros, dois intervalos de tempo, *data A* e *data B* e devolve o preço médio das viagens nesse dado intervalo de tempo, mas sem considerar gorjetas. Retorna os dados no seguinte formato:

1. preco_medio

O parâmetro que nos é devolvido, o preço médio das viagens, sem considerar gorjetas, baseia-se apenas no preço referente a viagens feitas no intervalo de tempo que fornecemos inicialmente.

Para a realização desta query, estudamos todas as viagens realizadas no espaço de tempo que oferecemos inicialmente, entre *data A* e *data B*, usando assim as datas específicas das *rides* aì realizadas, comparamos o preço consoante a classe do veículo de cada *driver*, quer seja, *Basic*, *Green* ou *Premium*. Só depois destes passos, calculamos a média e devolvemos o resultado final.

(1) => 0.580791s

(2) => 2.921s

(3) => 0.254113s

## 4.6 Query 6

Nesta query, recebemos três parâmetros, cidade e duas datas que definem um intervalo de tempo, *city*, *data A* e *data B*, respetivamente. Devolve depois a distância média percorrida nessa determinada cidade, durante esse intervalo de tempo. Retorna os dados no seguinte formato:

1. distancia_media

O parâmetro que nos é retornado, a distância percorrida, fica então dependente sempre da cidade escolhida e da possível existência de uma viagem feita dentro do intervalo de tempo que mencionamos.

Para a realização desta query nós iteramos pelo catálogo das viagens onde adicionamos a uma varíavel, *dist*, a distância percorrida nessa viagem se a viagem que estamos a ver nesse momento ocorreu na cidade passada como parâmetro e se ela ocorreu dentro do período delimitado por ambas as datas passadas como parâmetro, onde ao mesmo tempo vamos incrementando uma outra varíavel, $num_distances, para saber o nmero de viagens vlidas. A psessa avaliaons dividimos a dist pela$

## 4.7 Query 8

*Nesta query, recebemos dois parâmetros, o género, gender, e X que define o mínimo de anos dos perfis, ordenando depois de forma a que as contas mais antigas apareçam primeiro, mais especificamente, ordenar por conta mais antiga de condutor e, se necessário, pela conta do utilizador. Em caso de empate em todas as categorias, ordenar por id da viagem. Retorna os dados no seguinte formato:*

1. *id_condutor, nome_condutor, username_utilizador, nome_utilizador*

*Os parâmetros então devolvidos são, o id do condutor, o nome do condutor, o username do utilizador e o nome do utilizador.*

*Para a realização desta query primeiramente criamos um array de RIDE onde vai ser guardadas todas as viagens que cumpram os requisitos, de seguida analisamos se tanto o utilizador como o condutor se encontram com a conta ativa e após isso verificamos cada RIDE para procurar compatibilidade entre o género do utilizador e do condutor. Depois de verificar a mesma compatibilidade, nós ordenamos o array de cima de forma que seja listado os dados de acordo com a idade das contas encontradas e tudo aquilo proposto, começando pelo id, nome do condutor, username do utilizador e nome do utilizador. Caso haja empate, vamos para a conta do utilizador, se este empate persistir, ordenamos pelo id da viagem.*

*Tempo de execução – 2N - Linear.*

*(1) => 2.132474s e 2.189903s*

*(2) => 5.711s e 8.706s*

*(3) => 0.880739s e 1.175176s*

### 4.8 Query 9

*Nesta query, recebemos dois parâmetros, uma data A e uma data B que definem um intervalo de tempo, com o objetivo de listar as viagens nas quais o passageiro deu gorjeta. Ordenamos depois estes resultados por ordem de distância percorrida, em ordem decrescente. Caso haja um empate, as viagens mais recentes deverão aparecer primeiro, se este empate persistir, ordenar pelo id da viagem, também em ordem decrescente. Retorna os dados no seguinte formato:*

1. *id_viagem, data_viagem, distancia, cidade, valor_gorjeta*

*Os parâmetros devolvidos são o id da viagem, data da viagem, distância percorrida, cidade em questão e o valor da gorjeta oferecida.*

*Para a realização desta query, comparamos todas as rides existentes dentro desse intervalo de tempo que fornecemos à função. Depois iremos ver, dentro dessas rides estudadas, em quais é que foram oferecidas gorjetas. Depois disso, comparamos as distâncias percorridas, o id da ride em específico, a data dessa ride (dentro dos limites estabelecidos) e cidade onde tudo ocorreu.*

## 5 Conclusão

*O tempo de execução das queries encontra-se razoável, mas não ótimo, sendo que em algumas queries, por exemplo a 8, poderiam ter sido usadas outras estruturas de dados para tornar a ordenação das linhas de resultado muito mais eficientes. Porém achamos que da forma que se encontra o programa tem um tempo razoável por isso encontramo-nos satisfeito com ele.*

*Estatisticamente o tempo de criação de 3 tabelas com 1 milhões de linhas era entre 12 a 15 segundos mais 1 a 2 segundos para as libertar. Isto num computador razoável. A seguir dizemos algumas vantagens e desvantagens deste trabalho.*

1. *Desvantagens – Maior tempo de execução caso o número de comandos seja elevado.*
2. *Vantagens – Menor risco de o programa "arrebentar" por falta de memória, caso o número de linhas dos ficheiros seja muito, mas muito elevado; Tempo de execução normal caso os comandos sejam todos diferentes.Relativamente a problemas de memoria libertamos a memória alocada não necessária sempre que possível.*

*No geral, consideramos que este trabalho foi altamente influenciado pelo trabalho dos elementos André Alves e Gonçalo Freitas, mais confortáveis na matéria e muito mais empenhados, acompanhados pela ajuda do elemento Afonso Martins que se mostrou mais ativo nesta fase mais recente. Achamos que foi bastante interessante e útil a inserção dos módulos e das cápsulas, como mencionado anteriormente.*