Universidade do Minho
Escola de Engenharia

LABORÁTORIOS DE INFORMÁTICA lll

GRUPO 96

TRABALHO REALIZADO POR:

A100754 – Rafael Peixoto

A95485 – Miguel Carvalho

A100711 - João Rodrigues

INTRODUÇÃO

O principal objetivo deste projeto é o bom uso do conceito de modularidade e a aplicação do encapsulamento de dados.

O encapsulamento de dados é um mecanismo onde a implementação dos detalhes de uma classe é mantida em segredo dos outros módulos garantindo que a inexistência de um vazamento da estrutura de dados ou dos dados em si.

As vantagens do encapsulamento são a própria ocultação de dados/ implementações. Uma maior flexibilidade nas restrições nas estruturas de dados e existe uma maior facilidade na adaptação/mudança a novos requerimentos.

Modularidade consiste na subdivisão do programa em subprogramas separados, o que permite uma maior facilidade na alteração de módulos sem que seja preciso fazer alterações nos restantes.

Como utilizamos estes dois conceitos no nosso projeto?

Para tirar proveito do conceito de modularidade utilizamos os ficheiros (“user.c”, ”driver.c” e ”ride.c”) para o armazenamento de dados específicos a um membro singular para cada um destes campos e funções para a gestão dos mesmos membros.

Utilizamos os ficheiros (“users.c”, “drivers.c” e “rides.c”) para o armazenamento do conjunto dos membros singulares (sem conhecer a sua estrutura interna) e funções para a gestão destes catálogos de dados.

Utilizamos o programa “stats.c” para a coleção geral dos catálogos de dados e funções de gestão necessárias aos mesmos. Note que o programa “stats.c” não conhece os tipos dos catálogos que são guardados garantindo o encapsulamento.

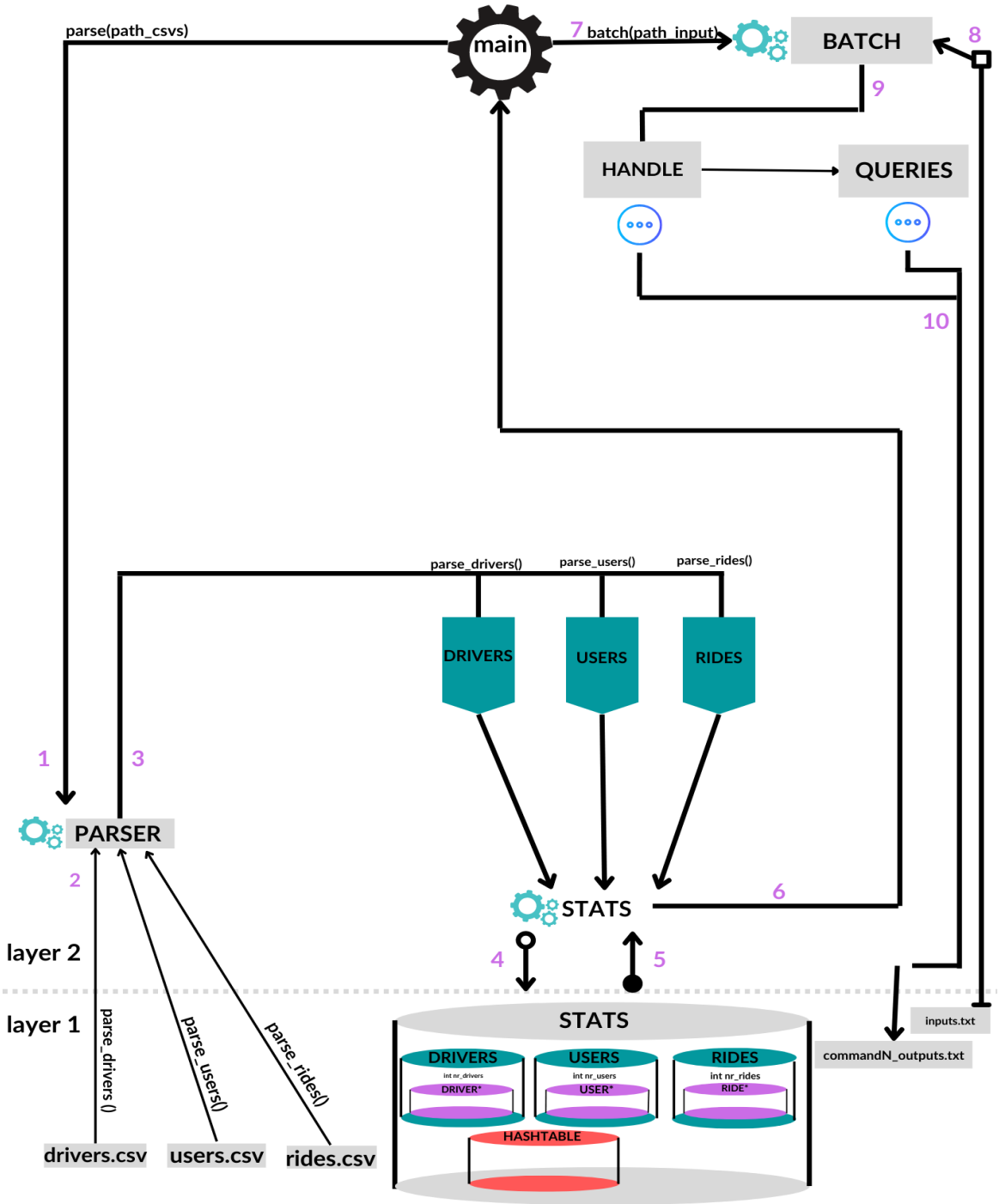
ESTRATÉGIA

“main.c”:

Este ficheiro é responsável pela geração do programa executável principal (“programa-principal”). E utiliza os seguintes módulos para a realização do mesmo:

- “parser.c”: recebe como argumento a localização da diretoria que contém os ficheiros .csv e retorna o conjunto de catálogos de dados no tipo Stats;
- “batch.c”: recebe a localização do input.txt e a Stats retornada pelo “parser.c” e é responsável pelo parsing de comandos (através do “handle.c”), execução (“queries.c”) e escrita dos outputs das queries (função write_output localizada em “queries.c”).

ARQUITETURA DO PROJETO



QUERY 4

Na query 4 o objetivo é devolver o preço médio das viagens (sem gorjeta) numa determinada cidade. A função recebe como argumentos a token_line que são os argumentos da linha de input dada, o catálogo das rides, o catálogo de drivers e o n que será o número de output.

A estratégia desta query foi utilizar uma função get_rides_by_city() onde se recebe o catálogo de rides, o catálogo de drivers e a cidade. Esta função percorre o catálogo das rides completo e compara a cidade dada com a cidade da ride se forem iguais irá se buscar o id do driver para obter a car_class para o cálculo do preço.

Depois de se calcular o preço da viagem acresce-se este valor ao preço_total das rides. No final basta apenas dividir o preço_total sobre o nº de rides que acontecem nessa mesma cidade e escrever o resultado para o output.

QUERY 5

Na query 5 o objetivo é determinar o preço médio entre duas datas (dataA e dataB) não incluindo o preço das gorjetas. São utilizados apenas o catálogo das rides e dos drivers

A estratégia utilizada para esta questão foi percorrer o catálogo das rides todas e a cada ride ver se a data está entre a dataA e dataB.

Se estiver entre as duas datas é adicionado o preço da viagem a um contador total do preço e é aumentado uma unidade o contador das viagens.

No final do ciclo for que lê todas as rides é calculado o preço médio com o preço total e com o contador das viagens.

Inicialmente aloca-se espaço para a dataA, dataB e para a data da Ride assim como para o resultado que será dado em string. Para isto usamos a função malloc.

Inicializamos o contador do nº de rides (nr_rides) entre as datas a zero e criamos um N que será o nº de rides do catálogo. Inicializamos também o contador do preço total (total_price) a zero.

Lê-se através do sscanf a token_line e guarda-se as datas dadas no input na dataA e na dataB respetivamente.

Inicia-se um ciclo for que irá percorrer o catálogo das rides completo, em cada ride guarda-se a data dela e verifica-se através de um if statement se está entre a dataA e a dataB , se estiver vai se buscar o id do driver com o get_ride_driver e através do id convertido para o index vai se buscar o id do driver com a função get_driver_by_id() com isto já será possível obter a car_class do driver pela função get_driver_car_class. Com estes dados calcula-se o preço da viagem e incrementa-se o total_price. Acrescenta-se também uma unidade ao nr_rides.

Acabando o ciclo calcula-se o preço médio dividindo o total_price pelo nr_rides.

Para dar o output passa-se o preço médio para string utilizando sprintf e dá-se o output do resultado com a função write_output.

QUERY 6

Nesta query o objetivo é calcular a distância média percorrida, numa determinada cidade entre dois intervalos de tempo.

A função "querie6" recebe a string "token_line", o catálogo das viagens "rs" e um inteiro "n".

Através da função "malloc" alocamos espaço para as strings denominadas por result (que será o output desta função), cidade (cidade que lemos nos inputs), dataA e dataB (datas entre as quais efetuaremos o cálculo).

Utilizamos também as variáveis "data_ride" (assumirá a data da ride que compararemos com as datas descritas anteriormente), "nr_rides" (que no inicio é inicializada a 0 e que no final terá o número total de viagens numa certa cidade num determinado tempo), "N" (que tem o valor do número total de viagens), "distancia_total" (que no inicio é inicializada a 0 e que no final terá a distancia total percorrida pelas viagens numa certa cidade num certo intervalo de tempo) e declaramos também a variável "dist_media".

Através da função sscanf lemos a token line e atribuímos as variáveis “cidade”, “dataA” e “dataB” os seus respetivos valores.

Através de um "for" que irá de 0 a N-1 (número de viagens - 1):

Na variável "r" guardamos a viagem com indice i;

Na variável "data_ride" guardamos a data da viagem "r";

Fazemos uma comparação entre a cidade guardada na variável "cidade" e a cidade da viagem guardada na variável "r". Se forem iguais (ou seja, mesma cidade) e se a "data_ride" se encontrar entre a dataA e a dataB: incrementamos a "distancia_total" com o valor da distância dessa viagem e aumentamos 1 ao "nr_rides".

No final do ciclo "for" calculamos o valor da distância média e guardamos na variável "dist_media" através da divisão entre "distancia_total" e "nr_rides".

Através da função "sprintf" guardamos na variável "result" o valor da variável "dist_media" com 3 casas decimais.

Através da função write_output escrevemos o output guardado na variável "result" no comandoN_output.txt

Como utilizamos "malloc" para "cidade", "dataA", "dataB" e "data_ride" no final dá-mos free a todas essas variáveis. Dá-nos também free ao catálogo das viagens "rs".

Análise de complexidade: O(N)

Onde N representa o número total de rides.

DIFICULDADES E MELHORIAS PARA A FASE 2

Uma das maiores dificuldades que tivemos neste projeto é a implementação de funções que libertam espaço para as estruturas de dados quando realizamos a cópia das estruturas (através da funcão strdup), para uma boa gestão de memória.

Outra das dificuldades que tivemos foi no armazenamento de rides de um user/driver para o acesso constante de todas as rides de um user/driver.