Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III

Carlos Diogo Fernandes Pina A95349
Lara Beatriz Pinto Ferreira A95454
João Machado Gonçalves A97321

Figura 1: A95349

Figura 2: A95454

Figura 3: A97321

18 de Novembro de 2023

Índice

1 Introdução

No âmbito da Unidade Curricular de Laboratórios de Informática III da licenciatura em Engenharia Informática, foi nos requisitado a implementação de um sistema em C que consiga dar resposta a *queries* pedidas, sobre a dados fornecidos em formato de texto. Na primeira fase foi nos requisitado a implementação de um *parser* em C que validasse as linhas dos quatro ficheiros *.csv* (*users*, *flights*, *passengers* e *reservations*) que nos foram disponibilizados, assim como a implementação de pelo menos seis *queries*, que no caso resolvemos implementar as *queries 1,3,4,5,8,9*. Nesta segunda e última fase, o objetivo era a implementação do modo interativo, dimensionar a solução para ficheiros de entrada com uma ordem de grandeza superior, completar totalmente as *queries* requisitadas e testar e avaliar o tempo de execução dessas mesmas *queries*. Para a realização do presente projeto, aplicamos os conhecimentos essenciais da linguagem C e Engenharia de Software, principalmente modularidade e encapsulamento.

2 Arquitetura do sistema

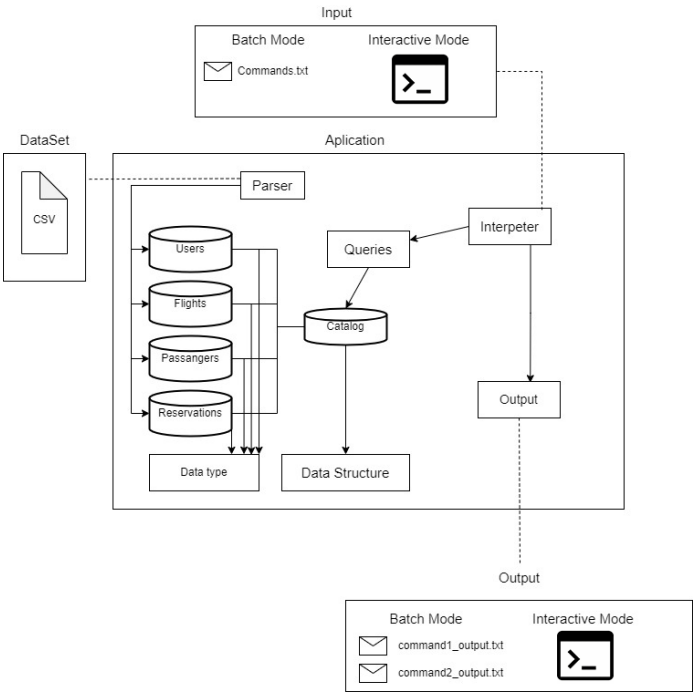

Figura 4: Arquitetura de Sistema

3 Estratégia utilizada

Após a análise do enunciado, decidimos adotar uma abordagem que envolve a criação de arquivos para armazenar dados válidos e a implementação de caches para otimizar os acessos a esses arquivos. Esta estratégia será aplicada às estruturas de dados *users*, *flights*, *reservations* e *passengers*, com a intenção de realizar operações iterativas de forma eficiente.

3.1 Estruturas de dados utilizadas para representar utilizadores, voos, reservas e passageiros

As estruturas de dados referidas foram criadas de forma a ser possível realizar, o mais eficientemente possível, tanto a nível de performance como de memória, as *queries* que implementamos.

3.1.1 Utilizadores

Após a criação da cache, fazemos o parsing,análise e validação da *dataset*,os *users* válidos são guardados num *file users_valid.csv*. Ao realizar uma pesquisa, a busca é primeiro realizada na cache antes de consultar o arquivo, utilizando o ID do usuário como chave. Essa abordagem visa otimizar a eficiência do processo, favorecendo o acesso rápido às informações mais recentes armazenadas em memória, antes de recorrer aos dados armazenados nos arquivos. Cada voo é representado no sistema pela *struct Users*, com as seguintes características:

```
struct users {
    char *id;
    char *name;
    char *email;
    char *phone_number;
    Date birth_date;
    char *sex;
    char *passport;
    char *country_code;
    char *adress;
    Datetime account_creation;
    enum pay_method pay_method;
    enum account_status account_status;

    int flights_total;
    int reservations_total;
    double spent_total;
};
```

Figura 5: Struct Users

- Id : O identificador do *user* é representado como uma *string*
- Name : O Nome do *user* é representado como uma *string*
- Email : O email do *user* é representado como uma *string*
- Phone_number : O número de telemóvel do *user* é representado como uma *string*
- Birth_date : A data de nascimento do *user* é representada como um *datetime*
- Sex : O género do *user* é representado como uma *string*
- Passaport : O número do passaporte do *user* é presentado como uma *string*
- Country_code : O código do país de residência
- Address : A morada do *user* é representado como uma *string*
- Account_creation : A data de criação da conta é representada com um *datetime*
- Pay_method : O método de pagamento é representado como um *enum*
- Account_status : O estado da conta é representado como um *enum*
- Flights_total : O número de voos realizados pelo *user* é representado como um inteiro
- Reservations_total : O número de reservas realizadas pelo *user* é representado como um inteiro
- Spent_total : O total de dinheiro gasto pelo *user* é representado como um *double*

3.1.2 Voos

Após a criação da cache, fazemos o parsing,análise e validação da *dataset*,as *flights* válidos são guardados num *file flights_valid.csv*. Ao realizar uma pesquisa, a busca é primeiro realizada na cache antes de consultar o arquivo, utilizando o ID do usuário como chave. Essa abordagem visa otimizar a eficiência do processo, favorecendo o acesso rápido às informações mais recentes armazenadas em memória, antes de recorrer aos dados armazenados nos arquivos. Cada voo é representado no sistema pela *struct Flights*, com as seguintes características:

```
struct flights {
    char *id_flights;
    char *airline;
    char *plane_model;
    int total_seats;
    char *origin;
    char *destination;
    Datetime schedule_departure_date;
    Datetime schedule_arrival_date;
    Datetime real_departure_date;
    Datetime real_arrival_date;
    char *pilot;
    char *copilot;
    char *notes;

    int total_passengers;
    int median_delay;
};
```

Figura 6: Struct Flights

- Id_flights : O identificador do *Flight* é representado como uma *string*
- Airline : A companhia aérea que realizou o *Flight* é representada com uma *string*
- Plane_model : O modelo do avião usado na *Flight* é representada com uma *string*
- Total_seats: O número de lugares total do avião usado na *Flight* é representada como um inteiro
- Origin : O aeroporto de origem da *Flight* é representada como uma *string*
- Destination : O aeroporto de destino da *Flight* é representada como uma *string*
- Schedule_departure_date : A data e hora planeada para a partida da *Flight* é representado por um *datetime*
- Schedule_arrival_date : A data e hora planeada para a chegada da *Flight* é representado por um *datetime*

- Real_departure_date : A data e hora efetiva da partida da *Flight* é representado por um *datetime*
- Real_arrival_date : A data e hora efetiva da *Flight* é representado por um *datetime*
- Pilot : O nomedo piloto que realizou a *Flight* é representado por uma *string*
- Copilot : O nome do Co-piloto que realizou a *Flight* é representado por uma *string*
- Notes : As observações sobre a *Flight* são representadas como uma *string*
- num_passengers : O número de passageiros que realizou a *Flight* é representado como um inteiro
- Delay : O atraso da *flight*, em segundos, é representado como um inteiro

3.1.3 Reservas

Após a criação da cache, fazemos o parsing, análise e validação da *dataset*,as *reservations* válidos são guardados num *file reservations_valid.csv*. Ao realizar uma pesquisa, a busca é primeiro realizada na cache antes de consultar o arquivo, utilizando o ID do usuário como chave. Essa abordagem visa otimizar a eficiência do processo, favorecendo o acesso rápido às informações mais recentes armazenadas em memória, antes de recorrer aos dados armazenados nos arquivos. Cada voo é representado no sistema pela *struct Reservations*, com as seguintes características:

```
struct reservations {
    char *id_res;
    char *user_id;
    char *hotel_id;
    char *hotel_name;
    int hotel_stars;
    int city_tax;
    char *adress;
    Date begin_date;
    Date end_date;
    int price_per_night;
    char *includes_breakfast;
    char *room_details;
    char *rating;
    char *comments;

    int nights; // numero de noites de uma reserva
    double price_res; // preco total de uma reserva
};
struct cache_reservations {
    GHashTable *reservations_cache;
    GQueue *reservations_queue;
    int capacity;
};
```

Figura 7: Struct Reservations

- Id_res : O identificador da *Reservation* é representado como uma *string*
- User_id : O identificador do *User* que fez a *Reservation* é representado como uma *string*
- Hotel_id : O identificador do hotel é representado como uma *string*
- Hotel_name : O nome do hotel é representado como uma *string*
- Hotel_stars : O número de estrelas do hotel é representado como um inteiro
- City_tax : A percentagem do imposto da cidade é representado como um inteiro
- Begin_date : A data de início da *Reservation* é representado como uma *date*
- End_date : A data de fim da *Reservation* é representado como uma *date*
- Price_per_night : O preço por noite é representado por um inteiro
- Includes_breakfast : A inclusão de pequeno-almoço na *Reservation* é representa como uma *string*
- Room_details : Os detalhes sobre o quarto são representado como uma *string*
- Rating : A classificação atribuída pelo *user* é representada como uma *string*
- Comment : O comentário sobre a reserva
- Nights : O número de noites que abrangidas na *Reservation* é representada como um inteiro
- Total_price : O preço total da *Reservation* é representado como um *double*

3.1.4 Passageiros

Após a criação da cache, fazemos o parsing,análise e validação da *dataset*,os *passengers* válidos são guardados num *file passengers_valid.csv*. Ao realizar uma pesquisa, a busca é primeiro realizada na cache antes de consultar o arquivo, utilizando o ID do usuário como chave. Essa abordagem visa otimizar a eficiência do processo, favorecendo o acesso rápido às informações mais recentes armazenadas em memória, antes de recorrer aos dados armazenados nos arquivos. Cada voo é representado no sistema pela *struct Flights*, com as seguintes características:

```
struct passengers {
    char *flight_id;
    char *user_id;
};

struct cache_passengers {
    GHashTable *passengers_cache;
    GQueue *passengers_queue;
    int capacity;
};
```

Figura 8: Struct Passengers

- Flight_Id : O identificador da *Flight* é representado como uma *string*
- User_Id : O identificador do *User* é representado como uma *string*

3.2 Estratégia de modularização e encapsulamento

Mantivemos a modularização e encapsulamento,pois estes promovem uma maior organização, leitura e facilidade de manutenção do código, que por sua vez auxiliam no processo de desenvolvimento, desta forma decidimos implementar os seguintes módulos:

- Users : Representação de um *User*, métodos de manipulação dos mesmos, a sua cache e metodos de manipulação da mesma
- Flights : Representação de um *Flight*, métodos de manipulação dos mesmos, a sua cache e metodos de manipulação da mesma
- Reservations : Representação de um *Reservation*, métodos de manipulação dos mesmos, a sua cache e metodos de manipulação da mesma
- Passengers : Representação de um *Passenger*, métodos de manipulação dos mesmos, a sua cache e metodos de manipulação da mesma
- Catalog : Agregação dos diversos catalogos e estruturas
- Date : Representação de um *Date* e/ou *Datetime*
- Valid : Agregado de métodos de validação
- Prints : Agregado de métodos de output(Batch e Menu Interativo)

4 Modo Interativo

Como requesitado, nesta segunda fase, para além do modo batch, no qual o programa é executado com dois argumentos (sendo o primeiro o caminho para a pasta onde estão os ficheiros de entrada e o segundo o caminho para um ficheiro de texto que contém uma lista de comandos a serem executados), implementamos também um modo interativo que recebe input do utilizador e apresenta as respostas às queries pedidas no terminal, usando um método de paginação.

5 Queries

5.1 Query 1

5.1.1 Descrição

Listar o resumo de um utilizador, voo, ou reserva, consoante o identificador recebido por argumento. É garantido que não existem identificadores repetidos entre as diferentes entidades. A query deverá retornar as seguintes informações:

- Utilizador : nome;sexo;idade;código_do_país;passaporte;número_voos;número_reservas;total_gasto
- Voo : companhia;avião;origem;destino;partida_est;chegada_est;número_passageiros;tempo_atraso
- Reserva : id_hotel;nome_hotel;estrelas_hotel;data_início;data_fim;pequeno_almoço;número_de_noites;preço_total

Não deverão ser retornadas informações para utilizadores com account_status = “inactive”.

5.1.2 Implementação

Se o identificador conter apenas dígitos, iremos procurar por uma *flights*, caso contrário, se os primeiros 4 caracteres forem '*Book*', procuraremos por uma *reservations*, caso nenhuma das condições anteriores se verifique, procuramos por um *users*. Todas as pesquisas são inicialmente conduzidas na cache correspondente ao dado procurado, sendo apenas em seguida consultados os arquivos respectivos. Em todos os casos é devolvida uma cópia dos valores encontrados ou nada se não existirem exceto no caso dos utilizadores com *account_status = “inactive”* que também não devolve nada.

5.2 Query 2

5.2.1 Descrição

Listar os voos ou reservas de um utilizador, se o segundo argumento for flights ou reservations, respetivamente, ordenados por data (da mais recente

para a mais antiga). Caso não seja fornecido um segundo argumento, apresentar voos e reservas, juntamente com o tipo (flight ou reservation). Para os voos, date = schedule_departure_date (contudo, deverá ser descartada a componente das horas/minutos/segundos no output), enquanto que para as reservas, date = begin_date.5 Em caso de empate, ordenar pelo identificador (de forma crescente). Tal como na Q1, utilizadores com account_status = "inactive" deverão ser ignorados.

5.2.2 Implementação

5.3 Query 3

5.3.1 Descrição

Apresentar a classificação média de um hotel, a partir do seu identificador.

5.3.2 Implementação

Procuramos todas as *reservations*, no arquivo das mesmas, que foram efetuadas no hotel com o identificador especificado e obtemos todas classificações. No fim, calculamos a media dessas classificações.

5.4 Query 4

5.4.1 Descrição

Listar as reservas de um hotel, ordenadas por data de início (da mais recente para a mais antiga). Caso duas reservas tenham a mesma data, deve ser usado o identificador da reserva como critério de desempate (de forma crescente).

5.4.2 Implementação

5.5 Query 5

5.5.1 Descrição

Listar os voos com origem num dado aeroporto, entre duas datas, ordenados por data de partida estimada (da mais recente para a mais antiga). Um voo está entre ¡begin_date¿ e ¡end_date¿ caso a sua respetiva data estimada de partida esteja entre ¡begin_date¿ e ¡end_date¿ (ambos inclusivos). Caso dois voos tenham a mesma data, o identificador do voo deverá ser usado como critério de desempate (de forma crescente).

5.5.2 Implementação

5.6 Query 6

5.6.1 Descrição

Listar o top N aeroportos com mais passageiros, para um dado ano. Deverão ser contabilizados os voos com a data estimada de partida nesse ano. Caso dois aeroportos tenham o mesmo valor, deverá ser usado o nome do aeroporto como critério de desempate (de forma crescente).

5.6.2 Implementação

5.7 Query 7

5.7.1 Descrição

Listar o top N aeroportos com a maior mediana de atrasos. Atrasos num aeroporto são calculados a partir da diferença entre a data estimada e a data real de partida, para voos com origem nesse aeroporto. O valor do atraso deverá ser apresentado em segundos. Caso dois aeroportos tenham a mesma mediana, o nome do aeroporto deverá ser usado como critério de desempate (de forma crescente).

5.7.2 Implementação

5.8 Query 8

5.8.1 Descrição

Apresentar a receita total de um hotel entre duas datas (inclusive), a partir do seu identificador. As receitas de um hotel devem considerar apenas o preço por noite (price_per_night) de todas as reservas com noites entre as duas datas. E.g., caso um hotel tenha apenas uma reserva de 100€/noite de 2023/10/01 a 2023/10/10, e quisermos saber as receitas entre 2023/10/01 a 2023/10/02, deverá ser retornado 200€ (duas noites). Por outro lado, caso a reserva seja entre 2023/10/01 a 2023/10/02, deverá ser retornado 100€ (uma noite).

5.8.2 Implementação

Ao receber como input o identificador do hotel e as datas de início e fim do período que queremos avaliar, a função verifica se as datas são válidas, de seguida procuramos todas as *reservations*, no arquivo das mesmas, que foram efetuadas num hotel e período especificados, calculamos o preço de cada *reservation* nesse período e somamos à receita total.

5.9 Query 9

5.9.1 Descrição

Listar todos os utilizadores cujo nome começa com o prefixo passado por argumento, ordenados por nome (de forma crescente). Caso dois utilizadores tenham o mesmo nome, deverá ser usado o seu identificador como critério de desempate (de forma crescente). Utilizadores inativos não deverão ser considerados pela pesquisa

5.9.2 Implementação

Procuramos todos os *users*, no arquivo dos mesmos, cujo nome começa com o prefixo especificado e a sua conta não está inativa, usamos-los para criar uma lista e organizamos essa mesma lista consoante os requisitos em cima.

5.10 Query 10

5.10.1 Descrição

Apresentar várias métricas gerais da aplicação. As métricas consideradas são: número de novos utilizadores registados (de acordo com o campo account_creation); número de voos (de acordo com o campo schedule_departure_date); número de passageiros; número de passageiros únicos; e número de reservas (de acordo com o campo begin_date). Caso a query seja executada sem argu- mentos, apresentar os dados agregados por ano, para todos os anos que a aplicação tem registo. Caso a query seja executada com um argumento, year, apresentar os dados desse ano agregados por mês. Finalmente, caso a query seja executada com dois argumentos, year e month, apresentar os dados desse ano e mês agregados por dia. O output deverá ser ordenado de forma crescente consoante o ano/mês/dia

5.10.2 Implementação

6 Teste de Performance

7 Conclusão