Universidade Do Minho
Licenciatura Em Engenharia Informática

LI3

Grupo 51

Diogo Ferreira (A104266) David Figueiredo (A104360)

Ano Letivo 2023/2024

Conteúdo

Capítulo 1

Introdução e desafios

Este projeto consistiu no desenvolvimento de um Sistema de Gestão de utilizadores, voos e reservas na linguagem de programação C. Sendo um projeto em grande escala foi necessário a utilização de estruturas de dados eficientes para armazenar e consultar grandes quantidades de informação. As estruturas de dados utilizadas no projeto foram criadas por nós ou pertencem à biblioteca GLib, no caso as Hash Tables.

O projeto encontra-se dividido em duas fases sendo que a primeira fase se concentra mais na implementação do parsing dos dados e no modo batch. Este modo envolve a consulta de dados armazenados num ficheiro de texto que é dado como argumento na execução do programa. Já na segunda fase para além dos requisitos da fase 1, adiciona-se agora o modo iterativo, testes funcionais de desempenho e memória, adequação do programa a uma base de dados maior, bem como a evolução de aspetos como modularidade e encapsulamento. É importante ter em consideração que a modularidade e encapsulamento são aspetos de extrema importância para o programa.

Capítulo 2

Módulos e estruturas de dados

2.1 Users

Para armazenar todos os users usámos uma hash table que associa uma chave a uma estrutura de dados que chamamos user. Um user permite armazenar, para um dado utilizador, todas as suas informações (id, nome, email, telemóvel, data de nascimento, sexo, passaporte, código do país, morada, data de criação de conta, método de pagamente e estado da conta), tendo assim a nossa definição de user:

```
typedef struct user{
    char * id;
    char * name;
    char * email;
    char * phone_number;
    struct time * birth_date;
    char sex;
    char * passport;
    char * country_code;
    char * address;
    struct time * account_creation;
    char * pay_method;
    bool account_status;
} User;
```

O módulo user permite realizar operações de inserção, pesquisa e atualização de dados.

2.2 Reservations

Para armazenar as reservations a estratégia utilizada é similar à estratégia usada para os users, sendo essa uma hash table, mas no caso das reservations associamos uma chave a uma estrutura de dados que chamamos de reservation, onde, tal como nos users, podemos armazenar todas as informações de uma reserva (id, id do utilizador, identificador do hotel, nome do hotel, estrelas do hotel, imposto da cidade em percentagem, morada do hotel, data de início, data de fim, preço por noite, inclusão de pequeno almoço, detalhes do quarto, rating, comentário sobre a reserva), tendo assim a nossa definição de reservation:

O módulo reservation permite realizar operações de inserção, pesquisa e atualização de dados.

```
typedef struct reservation{
    char * id;
    char * user_id;
    char * hotel_id;
    char * hotel_name;
    unsigned int hotel_stars;
    int city_tax;
    char * address;
    struct time * begin_date;
    struct time * end_date;
    double price_per_night;
    bool includes_breakfast;
    char * room_details;
    unsigned int rating;
    char * comment;
}Reservation;
```

2.3 Flights

Para armazenar os flights foi utilizada a mesma estratégia utilizada para os users e as reservations, utilizando uma estrutura de dados à qual chamamos flight onde podemos armezenar as informações de cada voo (id, airline, modelo do avião, total de lugares, origem, destino, data de partida estimada e real, data de chegada estimada e real, piloto, copiloto, observações dobre o voo), obtendo assim a nossa definição de flight:

```c
typedef struct flight{
    char * id;
    char * airline;
    char * plane_model;
    unsigned int total_seats;
    char * origin;
    char * destination;
    struct time * schedule_departure_date;
    struct time * schedule_arrival_date;
    struct time * real_departure_date;
    struct time * real_arrival_date;
    char * pilot;
    char * copilot;
    char * notes;
} Flight;
```

O módulo flight permite realizar operações de inserção, pesquisa e atualização de dados.

2.4 Passengers

Para armazenar os passengers, a estratégia utilizada foi diferente dos restantes dados, uma vez que para os passangers em vez de utilizar uma hash table optamos pela utilização de um array onde guardamos uma estrutura de dados que nomeamos passenger. Essa estrutura de dados permite armazenar um id de utilizador, que no caso é o passageiro, e o id do voo.

```c
typedef struct passenger{
    char * flight_id;
    char * user_id;
} Passenger;
```

O módulo passenger permite realizar operações de inserção, pesquisa e atualização de dados.

2.5 Estruturas Auxiliares

Para facilitar e tornar mais eficiente a resposta às queries definimos 3 estruturas de dados temporárias para armazenar dados necessários para a realização das queries. Tendo assim as estruturas Temporary, MultiRecord e SingularRecord.

```c
typedef struct temporary {
    void * database;
    char * id;
    void ** list;
    int num;
    int sum;
    void * begin;
    void * end;
```

```c
typedef struct mRecord {
    char ** names;
    int ** list;
    int size;
    int * listSize;
    short unsigned int * listMaxSize;
    int max;
} MultipleRecord;
```

```c
typedef struct sRecord{
    char ** names;
    int * list;
    int size;
} SingularRecord;
```

2.6 UI

O módulo ui, tal como o nome indica, refere-se ao modo interativo, ou a interface do utilizador. Esta API possui funções iniciar o modo interativo, para a impressão os outputs das queries na "janela", e outras para comodidades estéticas (cursor não aparece / aparece no ecrã, inputs são / não são escritos).

2.7 Interpreter

O módulo interpreter é o módulo que recebe o input do utilizador, identifica a querie correspondente e chama a função dessa querie que se encontra no módulo queries, fornecendo-lhe o input necessário para a realização da mesma. O modulo interpreter também é responsável por realizar um benchmark no qual é medido o tempo que cada querie demora a ser feita.

Capítulo 3

Dificuldades sentidas

Este trabalho não é de fácil execução sendo que a maior dificuldade nesta segunda fase foi a otimização das queries para conseguir reduzir o tempo de execução de modo a obter os tempos necessários para o dataset large, bem como otimizar de modo a diminuir a quantidade de memória utilizada. Outro dos grandes desafios foi a implementação da modularidade e encapsulamento uma vez que foi a primeira vez que o fizemos, tornando-se um grande desafio.

Capítulo 4

Testes de desempenho

Para medir o tempo de execução do nosso programa recorremos à biblioteca time.h, fazendo várias medições obtendo sempre medições similares à medição apresentada na seguinte tabela:

Queries	Regular Dataset	Large Dataset
1	0.001464 (36)	0.732924 (180)
2	0.005910 (12)	1.491383 (50)
3	0.008795 (6)	1.332406 (50)
4	0.010426 (6)	1.590361 (5)
5	0.000420 (6)	0.038112 (50)
6	0.010267 (6)	1.993365 (50)
7	0.000961 (4)	0.168071 (10)
8	0.010046 (10)	1.515738 (50)
9	0.002524 (8)	0.341914 (50)

Os tempos apresentados na tabela são divididos pelo número de vezes que a query foi feita, esse número está presente entre parêntesis ao lado do tempo de execução.

A query 10 não foi adicionada à tabela uma vez que não foi resolvida.

Com a análise da tabela podemos observar que a diferença dos tempos de execução do Regular Dataset para o Large Dataset é muito grande principalmente em queries que percorrem uma das database toda.

Para além dos testes de tempo de execução também foi realizado um teste de memória para ver a memória usada pelo programa, sendo esta 35584KB no Regular Dataset e 5640624KB no Large Dataset.

Capítulo 5

Conclusões

Concluindo, apesar das dificuldades que foram surgindo ao longo do projeto, acreditamos que tentamos ao máximo realizar um bom projeto que infelizmente não apresentou os resultados esperados pelos elementos envolvidos na realização do projeto uma vez que as queries não foram devidamente otimizadas fazendo com que algumas delas apresentem tempos de execução demasiado altos. Sentimos também que algumas estruturas e funções, apesar de funcionarem corretamente, poderiam ser mais simples. Quanto às estratégias de modularidade e encapsulamento que eram dos principais desafios na realização do programa, pensamos ter feito uma correta aplicação desses conceitos.

Grafo de dependências