Relatório da Fase 2 – LI3 – Grupo 13

Fernando Jorge da Silva Pires (a77399)
Edgar Carvalho Ferreira (a99890)
João Gomes Dias de Faria (a100553)

Janeiro 2024

Índice

1 – Introdução

Este relatório foi elaborado no âmbito da unidade curricular de Laboratórios de Informática III no ano letivo 2023/2024. Os objetivos da segunda fase são, para além das previamente definidas para a primeira fase, a implementação da totalidade das *queries*, o modo de operação interativo, testes funcionais e de desempenho, a evolução de aspetos como modularidade e encapsulamento, tudo isto adequado a um *dataset* de ordem de grandeza superior.

2 – Alterações relativas à primeira fase

Uma das maiores alterações reflete-se na criação de catálogos que agrupam as entidades (posteriormente abordadas) na tabela correspondente. Estes catálogos são iniciados no *main* e passados para o menu interativo e para o modo *batch*, previamente desenvolvido para a fase 1. Estas tabelas, na fase 1, estavam a ser criadas no modo *batch*, sendo que agora passaram para a *main*, pois também vão ser necessárias, e passadas como argumento, para o menu interativo.

Relativamente às *queries*, elas sofreram alterações que visam melhoria de desempenho.

A ordenação de dados pertencentes a alguns dos catálogos deixa de acontecer aquando da inserção dos mesmos, passando a ser ordenadas apenas quando necessárias, sendo chamadas nas *queries* que fazem uso deles. Este tópico será abordado tanto na parte das modelos de dados como nas *queries*.

3 – Desenvolvimento

3.1 - Modularidade, encapsulamento

No que toca a modularidade, procedemos à criação de módulos. Para tal, foram criadas as pastas ***include*** e ***src***. Dentro de cada uma delas temos diferentes módulos, como ***catalogs***, ***data_manager***, ***entities***, ***program***, ***queries, tests, stats*** e ***utils.***

Já o encapsulamento, que tem como principal razão a separação das interfaces e das respetivas implementações de forma que apenas se seja possível aceder à interface e não às

implementações, é garantido pela não possibilidade de acesso ao conteúdo das estruturas fora do ficheiro de implementação. Assim sendo, foram criados ***setters*** e ***getters*** para os campos do modelo de dados que seja pedido e catálogos das diferentes entidades.

3.2 – Memory leaks

No que toca a *memory leaks*, nesta fase, o programa apresenta 0.019Mb, valores que se devem à utilização da biblioteca *glib*, algo que abordaremos posteriormente.

3.3 - Execução inicial

No que toca à execução, a ***main*** cria os catálogos e trata da divisão entre ***batch_mode*** e o **menu interativo**. Caso seja recebido o caminho para os ficheiros CSV e os comandos necessários das *queries*, é iniciado o batch mode**,** caso não haja argumentos passados, é iniciado o menu interativo, explorado posteriormente.

No ***batch_mode***, são abertos todos os ficheiros *CSV* e os ficheiros das *queries*. Tudo isto é passado como argumento ao ***parser***, que tratará da leitura da informação dos ficheiros *CSV* e da respetiva inserção em cada um dos catálogos.

3.4 - Leitura e inserção de dados

Esta fase é desenvolvida e tratada pelo ***parser*** que tem como função ler os ficheiros CSV que contêm os dados para inserir em memória, todos lidos linha a linha e posteriormente validados no ***validater***.

Caso todos os campos da linha lida do *CSV* se encontrarem devidamente corretos de acordo com as informações dadas para a validação, a informação será condensada num dos modelos de dados e posteriormente adicionada aos catálogos que irão guardar tal informação em memória. Caso contrário serão escritos num ficheiro próprio para cada ficheiro de entrada.

3.5 - Queries

Esta fase é iniciada no ***batch_mode***, após o *parse* da informação e inserção nas tabelas. Aqui é chamada a função ***query_manager***, função implementada noutro ficheiro e que recebe como argumento todos os catálogos e o ficheiro com as queries. No ***batch_mode*** são fechados todos os ficheiros abertos.

Já no ***query_manager***, será obtida, linha a linha, a informação contida no ficheiro de inputs passado e tratada de forma a ser passada para cada uma das *queries* consoante o conteúdo. Cada *query* terá um ficheiro .c e .h, sendo que no .h são declaradas as funções e no .c está a implementação das mesmas.

- **Query 1 – Resumo de utilizador, reserva ou voo**

Esta *query* faz uso da rapidez de acesso a informação de cada *hashtable*. Verifica o id que é passado como input, identificando a qual dos modelos de dados corresponde e a existência. Após isso, escreve num ficheiro, o resumo consoante o pedido e de acordo com o previamente estabelecido de cada um dos modelos de dados (*user*, *reservation* e *flight*)

- **Query 2 – Listagem de voos ou reservas de um utilizador**

Esta *query*, contrariamente à anterior, pode receber mais que um argumento, sendo necessário tratar o input obtido. O primeiro argumento será o utilizador ao qual será necessário buscar a segunda parte do input, ou *flight* ou *reservation*, caso não seja passado um segundo argumento, serão apresentadas as duas hipóteses. Em ambos os casos, serão percorridas os *GArrays*, existentes no modelo de dados *users*, que contêm a informação tanto dos flights como das reservations que serão ordenados posteriormente.

- **Query 3 - Classificação média de um hotel**

A *query* 3 irá fazer uso do catálogo dos hotéis, por nós criado de forma a mais facilmente obter os resultados. O argumento dado será o *id* do hotel, chave da tabela dos hotéis, que servirá para obtermos, e escrevermos no ficheiro de resultado, o *rating* (ou seja, a classificação média) do hotel cujo id foi passado, algo calculado juntando todos os ratings e o número de ratings do catálogo dos hotéis que foi obtido através do ficheiro das *reservations*.

- **Query 4 – Listagem das reservas de um hotel**

Esta *query*, tal como a anterior, vai fazer uso do catálogo dos hotéis e das suas características. O argumento vai, novamente, ser o id do hotel cujas informações queremos obter. Será obtido esse hotel através da chave, o id, e o respetivo *GArray* das reservas desse hotel, sendo posteriormente ordenado e escritas, por ordem, as informações requeridas, ou seja: o *id* do hotel, a data de início e fim da reserva, o *id* do utilizador que fez tal reserva, a avaliação dada e o preço pago por noite.

- **Query 5 – Listagem de informações de voos de um aeroporto num intervalo de tempo**

Esta *query* vai fazer uso do catálogo dos aeroportos, mais propriamente do campo dos voos. Esta *query* recebe 3 argumentos, o nome do aeroporto, chave de cada entrada da tabela de voos, a data de começo e a data de fim. É necessário obter e ordenar o GArray que contém os voos de determinado aeroporto, não sendo contados os voos que não estão contidos naquele intervalo de tempo, sendo posteriormente escrito num ficheiro, o id do voo, o tempo de partida agendado, o destino, a companhia aérea e o modelo do avião.

- **Query 6 – Listagem do top de aeroportos**

Esta *query* vai utilizar novamente o catálogo dos aeroportos. A entidade aeroporto tem como atributo uma *GHashTable* que contém, para um ano, o número de passageiros desse aeroporto. Serão colocados num *array* cada um dos aeroportos, sendo posteriormente ordenados de acordo com o número de passageiros por ano, ano esse passado como argumento da *query*. Com o *array* ordenado, serão escritos no ficheiro os N, também passado como argumentos, aeroportos com maior número de passageiros.

- **Query 7 – Listagem de aeroportos com as N maiores medianas de atraso**

A *query* 7 irá fazer uso da tabela dos aeroportos e das suas características, incidindo no campo do atraso mediano. O atraso de cada voo é calculado a partir da diferença entre a hora estimada e a real de partida de cada voo e colocado num atributo da entidade representativa do voo. Estes valores serão posteriormente utilizados de forma a ser calculada a mediana de atraso. Tendo essa informação inserida no catálogo de aeroportos, ela será percorrida de forma a obter os N (valor passado como argumento) aeroportos com a maior mediana de atrasos, sendo escrito o nome e a mediana no ficheiro de saída.

- **Query 8 – Receita de um hotel num intervalo de tempo**

Esta *query* vai fazer uso da tabela dos hotéis e das suas características. Os argumentos são o id do hotel cujas informações queremos obter, uma data de começo e uma de fim. Será obtido esse hotel através da chave, o id, percorrida o *GArray* e, consoante a data de início e fim da reserva, calculado o valor pago nessa reserva do hotel. Esse valor será somado a uma variável que será o resultado da *query* e escrita no ficheiro.

- **Query 9 – Listagem de utilizadores cujo nome começa com o prefixo dado**

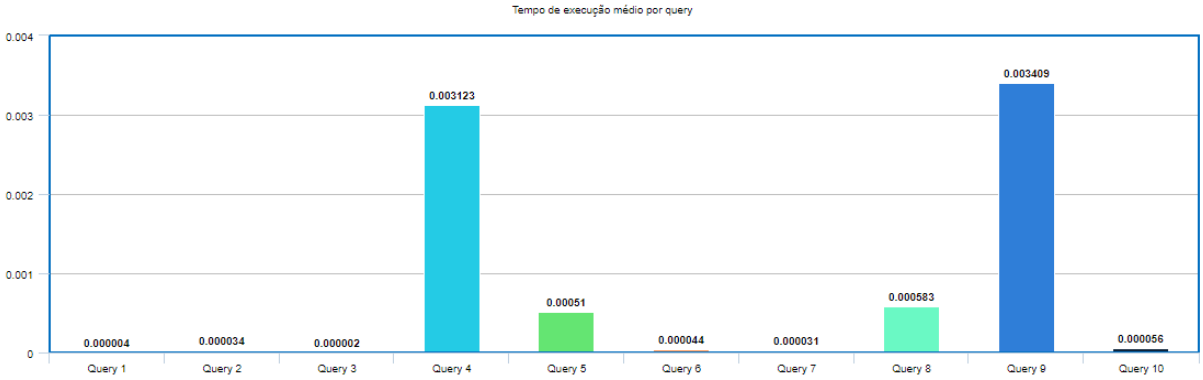
A *query* 9 incide sobre a tabela de utilizadores. Esta será percorrida de forma a verificar, entrada a entrada, se o argumento passado é, ou não, prefixo do nome de cada uma das entradas. Posteriormente, estes valores serão ordenados por ordem alfabética e, em caso de empate, pelo valor de id de cada um. Os resultados (*id* e nome do utilizador) são, posteriormente, colocados num ficheiro.

- **Query 10 – Apresentação de métricas gerais**

Esta *query* vai fazer uso de uma estrutura de métricas, previamente calculadas na inserção de dados. Os argumentos podem ser o ano e o mês, em que damos o output diário, apenas o ano em que damos o output mensal ou nenhum, em que damos o output anual de todos os anos que a aplicação tem registo. Para escrever os resultados apenas fazemos uma pesquisa direta na estrutura de métricas, caso seja diária ou verificamos todos as estruturas diárias caso seja o mês ou verificamos todas as estruturas mensais caso seja o ano. Quanto aos passageiros únicos, todas as estruturas têm uma *GHashTable* que contém os passageiros únicos sendo que depois vamos buscar apenas o tamanho desta tabela.

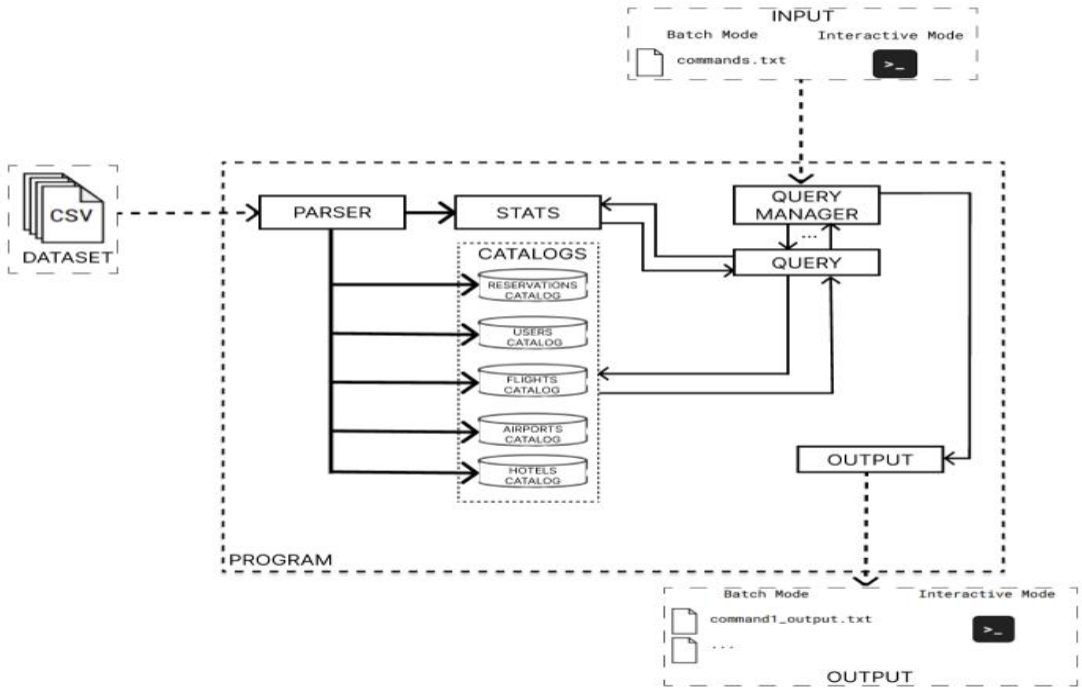
- **Resultados por execução de *query***

Estes valores foram obtidos após 20 execuções de cada *query*, sendo os resultados apresentados em segundos. Foi utilizado um computador com o OS ***6.6.1-Arch1-1***, com ***7.56GiB*** de memória RAM e com o processador ***Intel i5-1135G7, 11th Gen @ 4.200GHz***



3.6 - Estrutura geral

Graficamente, quanto a estrutura geral do projeto, podemos dividir da seguinte maneira



4 – Modelo de dados

4.1 – Catálogos

- **Reservation**
  Esta estrutura serve para armazenar informações relativas a reservas de hotéis. Comparativamente à primeira fase, valores que podem ser convertidos em inteiros, passaram a esse tipo de forma a fazer melhor gestão e uso de memória. Os atributos desta entidade foram criados como forma de suprimir as necessidades de algumas *queries*, de forma a otimizar a sua execução.
- **User**
  Esta estrutura serve para armazenar informações relativas a utilizadores. Comparativamente à primeira fase, da mesma forma da entidade *reservation*, valores que são passíveis de ser convertidos para inteiro foram, principalmente dos campos de *flight_list* e *reservation_list*, de forma a ocupar menos memória. Estes campos, na fase 1, encontravam-se previamente ordenados, algo que notamos não ser necessário e que tornavam o programa menos eficiente em termos de tempo. Assim sendo, estes atributos como são necessários para *queries*, são ordenados quando chamados, evitando assim ordenações desnecessárias para a execução.
- **Flight**
  Esta estrutura serve para armazenar informações relativas a voos. Esta estrutura teve a remoção da lista de passageiros e a maioria dos atributos passaram a inteiros (ou *longs* caso se ultrapasse a capacidade), pelo mesmo motivo das entidades anteriores. Além disso, há campos como o *delay* e o *passenger_numbers* que foram mantidos devido à utilidade que têm para algumas das *queries*.
- **Hotel**
  Esta estrutura serve para armazenar informações relativas a hotéis. A sua criação mostrou se de extrema importância para algumas das *queries*, diminuindo substancialmente o tempo de execução das mesmas. Tal como na entidade de *user*, na fase 1 esta estrutura tinha um *GArray* ordenado de reservas, sendo que foi mudada de forma a guardar inteiros e apenas é ordenado caso haja tal necessidade nas *queries*.
- **Airport**
  Esta estrutura serve para armazenar informações relativas a aeroportos, sendo útil para algumas das *queries* deste programa. Relativamente à fase 1, foi adicionado o atributo de *flight_list*, um *GArray* de inteiros, útil para o cálculo de valores como o campo da mediana de atraso do aeroporto. Outro dos atributos úteis para as *queries* é a *GHashTable* dos passageiros anuais.

4.2 - Utilização de bibliotecas

Sendo possível a inclusão da biblioteca ***glib*** no projeto, as suas estruturas de dados já otimizadas foram o maior atrativo para a sua utilização. Além destas, já se encontravam predefinidas funções de manipulação das tais estruturas, algo que nos facilitou a transformação do nosso processo de desenvolvimento mental do projeto em código.

Para o menu interativo, posteriormente abordado, utilizamos a ***ncurses***, biblioteca que nos deixa manipular a aparência do terminal de forma a tornar o uso mais intuitivo deste menu.

5 – Menu interativo

Esta parte do projeto é iniciada quando utilizador chama o programa sem argumentos adicionais. O terminal passará a mostrar uma tela onde é pedido o *path* para o *dataset* que servirá para povoar os catálogos, utilizando o *parser* e o *validater* previamente desenvolvidos. Se no destino desse *path* se encontrarem os ficheiros necessários para o povoamento dos catálogos, serão mostradas telas com as diferentes *queries* possíveis de executar. O utilizador poderá navegar neste menu utilizando as setas do teclado, passando de tela de *query*, onde irá ter uma breve descrição das *query*. Caso decida avançar, premindo o botão *Enter*, terá uma descrição mais detalhada da *query* e um exemplo do *input* pedido para a executar. O *input* é tratado e validado e, caso tudo esteja de acordo com o esperado, a *query* irá ser executada e criado o ficheiro de resultado. Esses resultados serão lidos dos ficheiros e impressos na tela, com divisão por páginas navegáveis em caso de necessidade. O utilizador poderá regressar à tela das *queries* e executar outra que deseje até decidir encerrar este modo.

6 – Testes funcionais e de desempenho

Para esta parte do projeto podemos começar por falar por do que são os testes. Estes são uma estrutura que armazena um *array* de *Queries*, representando um conjunto de testes a serem executados. Durante a execução no *test_manager*, cada *query* é submetida, e são registados os seguintes dados para cada execução:

- Tempo de Execução: Calculado para cada *query*, representando o tempo decorrido desde o início até a conclusão da operação.
- Resultado: Indica se a execução foi bem-sucedida ou não.
- Incongruência: Regista o ponto exato onde a saída diverge do resultado esperado, proporcionando informações detalhadas para análise posterior.

Ao final da bateria de testes, é gerado um resumo consolidado, incluindo o tempo total de cada *query*, o tempo médio de cada *query*, o número total de *queries* executadas e quantas produziram os resultados corretos, sendo que também são representadas numa percentagem de *queries* corretas em relação ao total. Além do resumo dos testes, são apresentadas estatísticas adicionais, tais como o tempo de *parser*, ou seja, o tempo gasto na leitura dos dados e na preparação para execução, o tempo total de todas as *queries*, o tempo total do programa, como soma das duas anteriores e a memória consumida durante a execução dos testes.

7 – Conclusões

Em jeito de conclusão, podemos afirmar que este projeto ajudou a estabelecer conceitos como encapsulamento e modularidade, além de nos incentivar à exploração de bibliotecas e suas funcionalidades numa linguagem como o C. Características como o controlo direto sobre o hardware e manipulação de apontadores tornam esta linguagem bastante boa para perceber melhor temas como o performance e gestão de memória.

Apesar disto, trabalhar com C pode resultar em alguns problemas relacionados com *memory leaks* algo que foi difícil de lidar durante o desenvolvimento do projeto.

Um dos maiores desafios deste projeto passou pela otimização das *queries*, sendo que nos focamos em ter o melhor desempenho possível nestas, tornando a leitura e inserção de dados mais lenta e em estruturas que melhor respondessem às necessidades. Relativamente a execução com *dataset* de grandeza superior obtivemos os seguintes resultados.