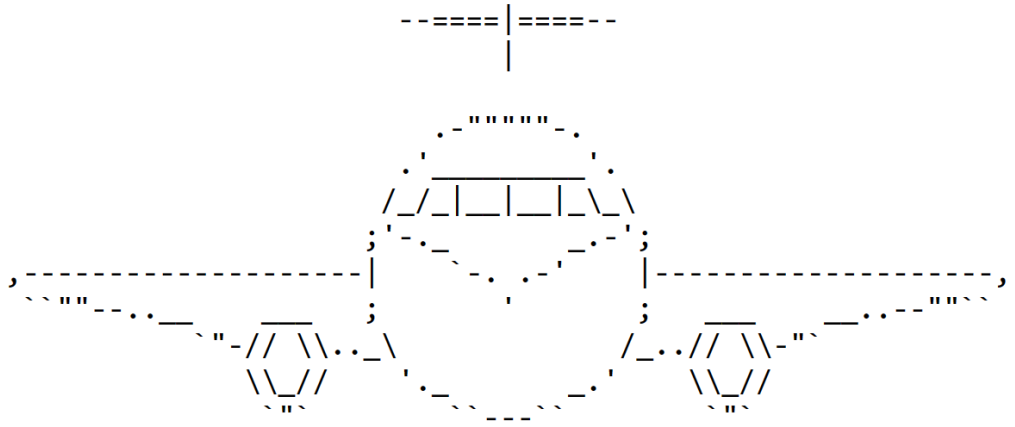


Relatório 2ª Fase | Laboratórios de Informática III
Grupo 1 | 2023/2024

Hélder Gomes (A104100) **João Lobo (A104356)** **Rita Camacho (A104439)**

Índice

1. Introdução

O presente relatório apresentará informações relativas à 2ª Fase do Projeto Final da Unidade Curricular Laboratórios de Informática III, pertencente ao 2º Ano da Licenciatura em Engenharia Informática, realizada no ano letivo 2023/2024, na Universidade do Minho.

Concluída a 1ª fase, na qual foi implementado o *parsing* e validação dos dados, o modo *batch* e 6 das 10 *queries* apresentadas pela equipa docente, completámos os restantes requisitos previstos na 2ª e final fase: conclusão das 4 *queries* restantes, criação do modo interativo, e testes funcionais e de desempenho.

2. Desenvolvimento

Para o projeto final, conseguimos implementar todos os modos descritos no guião (*batch* e interativo), bem como todas as *queries*, e ainda uma aplicação secundária de testes e cálculo de estatísticas de *performance* da aplicação, mantendo a qualidade de código já seguida na 1ª fase, documentação do projeto na totalidade, encapsulamento e modularidade. Através de pesquisa feita pelos elementos do grupo à documentação da *GLib* e a métodos algorítmicos de organização e pesquisa de dados, conseguimos tirar proveito de diversas funções e estruturas para obter uma aplicação final bastante eficiente.

2.1. Ajustes Realizados

Seguem-se algumas alterações executadas no desenvolvimento da 2ª fase, tanto por aconselhamento dos docentes durante a 1ª defesa, como para funcionamento ideal do programa com o novo *dataset* de um tamanho razoavelmente maior ao inicialmente usado.

2.1.1. Indicações dos Docentes

Foram-nos sugeridos alguns métodos de otimização pelos docentes, durante a defesa da 1ª fase, para aperfeiçoarmos o relatório e o desempenho de algumas *queries*, tendo sido aplicadas as seguintes sugestões:

- Aperfeiçoamento de gráficos presentes no relatório (tornando-os mais explícitos e completos);
- Implementação de *binary search* na *query* 5, onde há delimitação de datas.

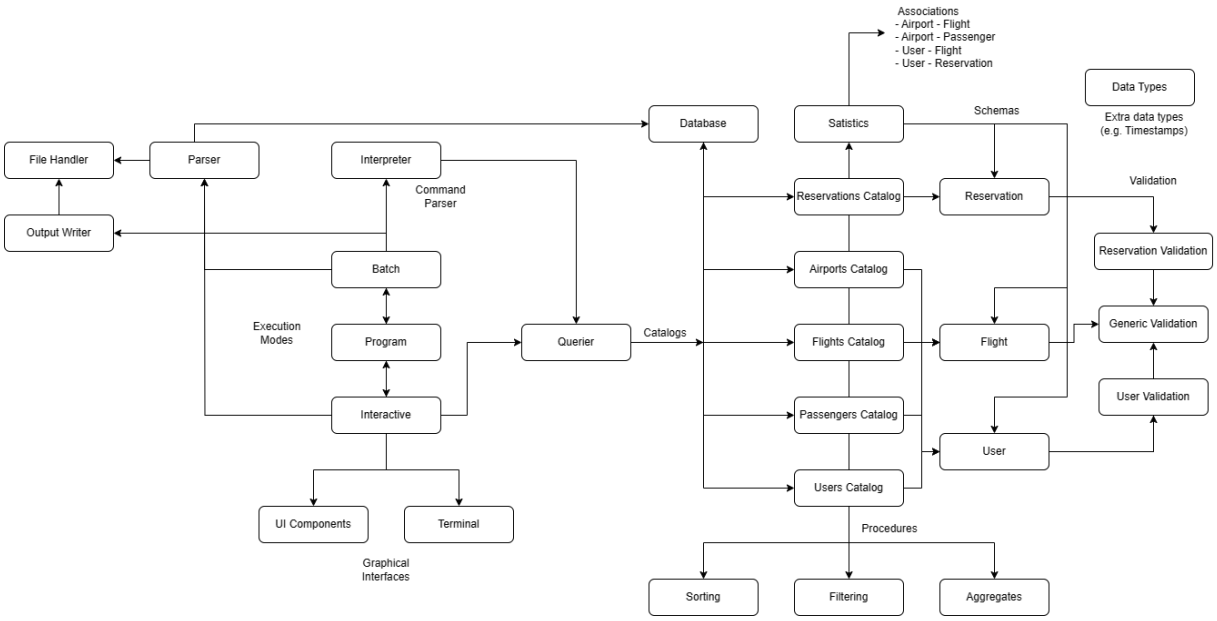
2.1.2. *Large Dataset*

O nosso projeto não teve quaisquer problemas a executar o *dataset* de maior tamanho logo na 1ª vez em que o mesmo foi introduzido no sistema de testes (*li3.di.uminho.pt*), no entanto, o programa sofreu as seguintes modificações, para que alcançasse o funcionamento mais ideal:

- Parâmetros de estruturas reescritos para utilizar tipos de dados equivalentes que ocupem menos memória;
- Utilização da função *quick sort* como alternativa à função de *sorting* pertencente à *GLib* (*g_ptr_array_sort*).

2.2. Arquitetura Aplicada

Este projeto foi desenvolvido através de vários módulos relacionados entre si:



2.2.1. Estruturas de Dados

Para realizar a implementação das *queries* restantes, foi necessária a adição de diversas estruturas de dados, de forma a aproveitar ao máximo a biblioteca *GLib*, e obter a melhor relação tempo-memória por *query*. Essencialmente, foi criado um novo catálogo de aeroportos para facilitar o desenvolvimento das *queries* 6 e 7 (as restantes alterações e inserções a catálogos já existentes são descritas no capítulo das *queries*).

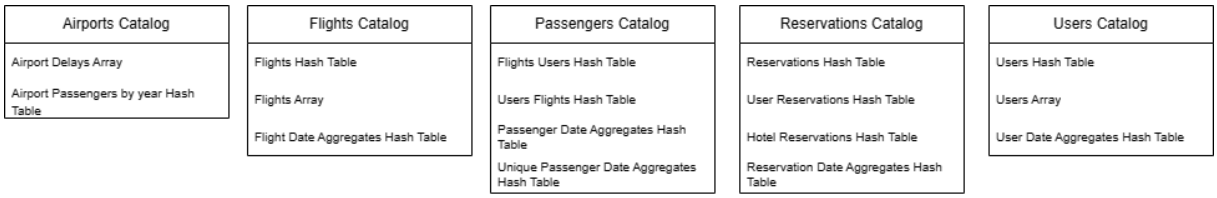


Figura 1: Estruturas de dados utilizadas em cada catálogo

No contexto do projeto, as *arrays* e as tabelas de *hash* utilizadas são, respetivamente, as *GPtrArray* e *GHashTable*, disponibilizadas pela *GLib*. Sendo assim, a informação e memória alocada para uma dada entidade não é repetida, apenas os *pointers* são organizados de formas diferentes.

2.3. *Queries*

Das 10 *queries* apresentadas, foram concluídas as 4 restantes na 2ª fase:

2.3.1. *Query* 6

Lista o top N aeroportos com mais passageiros, para um dado ano. Esta *query* é realizada através de dois passos realizados em momentos distintos: primeiramente, durante a inserção dos passageiros e através do módulo *statistics*, estes são quantizados e inseridos numa tabela de *Hash* que relaciona aeroportos com anos. Após a inserção ocorrer, é criada uma tabela de *Hash* que relaciona anos com uma *GPtrArray* de número de passageiros de um dado aeroporto, ordenada de forma crescente. Através destes dois procedimentos, a execução da *query* torna-se eficiente, necessitando apenas de obter a *array* de um respetivo ano e "extrair" os seus N primeiros elementos.

2.3.2. *Query* 7

Lista 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 é apresentado em segundos. Tal como a *query* 6, funciona através de dois procedimentos: inicialmente, durante a inserção dos voos, são gravados todos os atrasos numa *GPtrArray*, associada ao respetivo aeroporto numa tabela de *Hash*, presente no catálogo dos aeroportos, através de uma associação realizada pelo módulo *statistics*. Depois das inserções, é realizado um *quick sort* por atraso de forma crescente, e é calculada a mediana de atrasos por aeroporto, sendo posteriormente adicionados ordenadamente a uma *GPtrArray* de medianas de atrasos de aeroportos, ordenada de forma crescente. Durante a execução da *query*, facilmente se obtêm os top N aeroportos, selecionando os N primeiros elementos desta *array*.

2.3.3. *Query* 8

Apresenta 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. Através de uma tabela de *Hash* que relaciona um hotel com uma respetiva *GPtrArray* de reservas, a implementação desta *query* foi trivial, via um *quick sort* e um somatório do lucro por reserva.

2.3.4. *Query* 10

Apresenta 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). No caso desta *query*, para guardar a informação pedida (contagem das entidades por data), adicionámos uma tablela de *Hash* por catálogo para realizar a associação entre *timestamps* e inteiros.

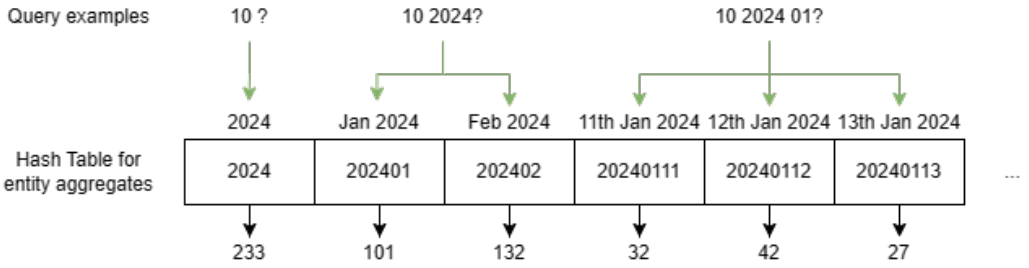


Figura 2: Exemplo da tabela de *Hash* existente para cada métrica

No caso dos passageiros únicos foi ainda necessário manter numa tabela secundária os utilizadores já adicionados ao somatório para evitar repetições.
Através destas abordagens, foi possível alcançar uma relação tempo-memória bastante eficiente para uma *query* de cariz complexo.

2.4. Modo Interativo

O modo interativo (interface gráfica no terminal), um dos requisitos da 2ª fase, permite que o utilizador forneça ao programa a diretoria dos ficheiros com os dados e realiza *queries* introduzidas pelo mesmo. Utilizámos a biblioteca *ncurses*, já conhecida anteriormente por nós, para realizar o mesmo, simplificando a implementação de um modo gráfico. O mesmo é capaz de:

- Listar uma breve descrição da *query* no mesmo menu onde se inserem os *inputs* para a mesma;
- Processar *queries* e devolver os seus resultados;
- No caso dos resultados serem demasiado extensos para imprimir a sua totalidade simultaneamente no ecrã, os resultados são paginados e o utilizador pode ler página a página os mesmos.

A implementação deste modo foi desenvolvida de forma modular e tendo em consideração a qualidade de código do projeto, através da criação de componentes de *UI* reutilizáveis como *modals*, menus e texto.

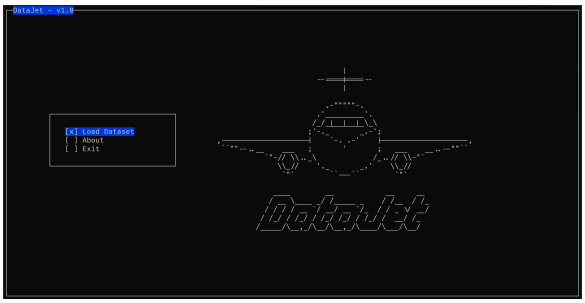


Figura 3: Menu Principal

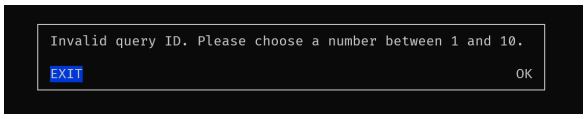


Figura 4: *Pop up* aviso para quando há indicação de argumentos impossíveis

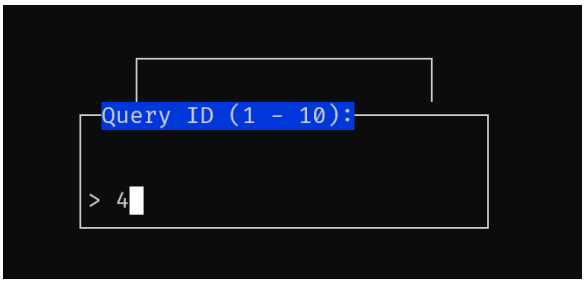


Figura 5: Menu para indicação da *query* desejada

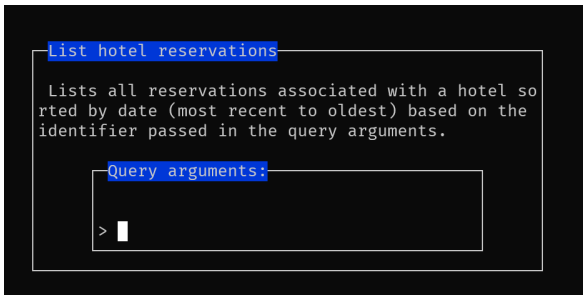


Figura 6: Menu para indicação dos argumentos necessários para a *query* escolhida

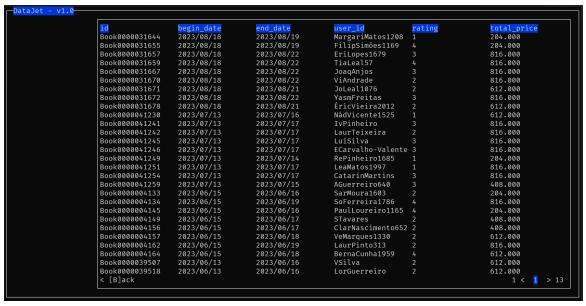


Figura 7: Página com impressão de resultados de uma *query*

Figura 8: Paginação dos resultados das *queries*

2.5. Testes Funcionais e de Desempenho

Desenvolvemos um módulo de testes para validar a correta implementação das *queries*, bem como o desempenho do programa. As métricas analisadas por este sistema são o tempo de execução das diferentes etapas de execução, o pico de memória utilizada e a correção e validação dos *outputs* do programa. Os dados gerados pelo módulo de testes são apresentados no terminal de forma gráfica, recorrendo a funções de *formatting* de *outputs*.

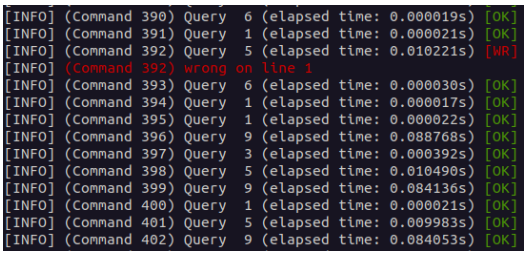


Figura 9: Correção de Comandos

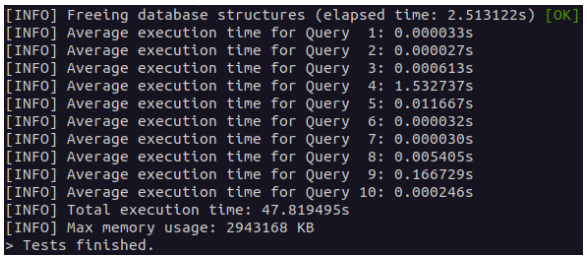


Figura 10: Métricas

2.6. Análise de Desempenho

A performance do projeto foi analisada utilizando os nossos computadores:

- Dispositivo 1 (LG Gram 15Z90Q-H.AP79P)
  - Especificações: 12th Gen Intel® Core™ i7-1260P 2.1GHz, 16GB RAM LPDDR5 5200 MHz (Ubuntu 22)
- Dispositivo 2 (LG Gram 17Z90Q)
  - Especificações: 12th Gen Intel® Core™ i7-1260P 2.1GHz, 16GB RAM LPDDR5 5200 MHz (Windows WSL2 Ubuntu 22)
- Dispositivo 3 (Lenovo ThinkPad T480s)
  - Especificações: 8th Gen Intel® Core™ i7-8650U 1.9GHz, 16GB RAM (Ubuntu 22)

	Dispositivo 1	**Dispositivo 2**	**Dispositivo 3**
Query 1	0.000043s	0.000163s	0.000092s
Query 2	0.000034s	0.000134s	0.000092s
Query 3	0.000674s	0.002110s	0.000900s
Query 4	1.997196s	1.544992s	1.374499s
Query 5	0.013997s	0.014568s	0.042182s
Query 6	0.000064s	0.000208s	0.000092s
Query 7	0.000039s	0.000142s	0.000093s
Query 8	0.006052s	0.008471s	0.009451s
Query 9	0.204541s	0.182522s	0.276739s
Query 10	0.000246s	0.000338s	0.000295s
Tempo de Inserção	40.950376s	44.337614s	41.574053s
Tempo de Execução Total	52.395921s	62.495581s	64.938225s
Pico de Memória Utilizada	2943216 KB	2944204 KB	2943868 KB

Tabela 1: Tempo de execução médio das 10 *queries*, inserção, execução total e máximo de memória usada

2.7. Otimizações

Como referido anteriormente, após a disponibilização do *large dataset* pela equipa docente, decidimos implementar algumas otimizações para melhorar a *performance* da aplicação.

Principalmente, alterámos alguns tipos das estruturas de dados para tipos equivalentes que gastem menos memória, por exemplo, utilizando *shorts* em vez de *ints* para números pequenos. Passámos a utilizar a função de *quick sort* da *standard library* de C pois, através de alguns testes, descobrimos que é mais eficiente que a disponibilizada pela *GLib* (*g_ptr_array_sort*). Aproveitamos então para criar um módulo responsável por este tipo de procedimentos (*sorting*).

2.8. Estratégias de Encapsulamento

Como referido no guião, um dos objetivos principais do projeto final e, consequentemente, da U.C., é a aplicação de estratégias de encapsulamento no seu desenvolvimento. Durante todo o progresso do projeto, tivemos em conta este ponto e aplicámos alguns dos seguintes conceitos:

- Utilização de estruturas opacas: a definição das estruturas do projeto foi maioritariamente escrita nos ficheiros *.c* e as declarações nos *.h*, limitando então o acesso aos campos das estruturas às funções de acesso, implementadas no módulo;
- Foram implementadas funções de *getters* e *setters* dentro dos módulos para criar e manter uma camada de proteção de acesso das estruturas e mantê-las opacas.

2.9. Dificuldades Sentidas

Como alcançámos uma ótima base na 1ª fase, a 2ª e última fase foi mais acessível, não existindo grandes dificuldades no desenvolvimento a apontar.

3. Conclusão

Concluindo, o desenvolvimento da 2ª Fase foi realizado da melhor forma possível, especialmente pela ótima base obtida já na 1ª Fase, encontrando-se alcançados todos os objetivos apresentados para a conclusão do projeto final da U.C. Laboratórios de Informática III.