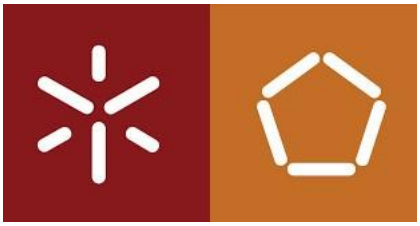


Universidade do Minho

Licenciatura em Engenharia Informática

Laboratórios de Informática III

Relatório 2ª Fase | Grupo 4

Diogo Ribeiro (A105995) Diogo Campos (A106920) Juliana Silva (105572)

Ano Letivo 2024/2025

ÍNDICE

Pág.

1. Introdução

No âmbito da unidade curricular de *Laboratórios de Informática III*, foi-nos proposto o desenvolvimento de um projeto em linguagem C, dividido em duas fases distintas.

A primeira fase teve como objetivo a criação de três *queries*, permitindo-nos explorar diversas ferramentas e adquirir boas práticas de programação. Estas competências foram fundamentais para a execução da segunda fase do projeto.

Na segunda fase, foi necessário realizar alterações nas *queries* previamente desenvolvidas, bem como implementar três novas *queries*, cada uma com objetivos distintos. Esta etapa teve como foco principal a aplicação de princípios de encapsulamento e modularização, considerados essenciais para um código bem estruturado e eficiente. Adicionalmente, os *datasets* utilizados foram substituídos por versões de dimensão significativamente superior, exigindo um tratamento mais robusto e eficiente dos dados.

Por fim, foi desenvolvido um modo interativo, proporcionando uma experiência mais dinâmica e intuitiva para a utilização do programa.

2. Desenvolvimento

Neste projeto, conseguimos concretizar com sucesso os desafios propostos no guião. Iniciámos esta fase aplicando as críticas construtivas recebidas por parte dos docentes, o que nos permitiu aprimorar significativamente o encapsulamento e a modularização do código, facilitando a transição para esta nova etapa.

Ao longo do desenvolvimento, o nosso grupo estabeleceu metas claras, superando as dificuldades que surgiram ao longo do caminho. Conseguimos implementar corretamente todas as *queries*, respeitando as restrições de utilização de memória e os tempos definidos pela equipa docente. Para reforçar a verificação destas competências essenciais, desenvolvemos um programa de testes que nos permitiu avaliar e validar a eficiência e a funcionalidade do projeto.

Além disso, realizámos a validação das *queries*, assegurando a sua robustez e precisão, e implementámos o modo interativo do programa, que enriqueceu a usabilidade e a experiência do utilizador. Outra etapa importante foi a otimização do desempenho do programa, ajustando algoritmos e estruturas de dados de modo a garantir uma execução mais eficiente, especialmente com o aumento dos datasets.

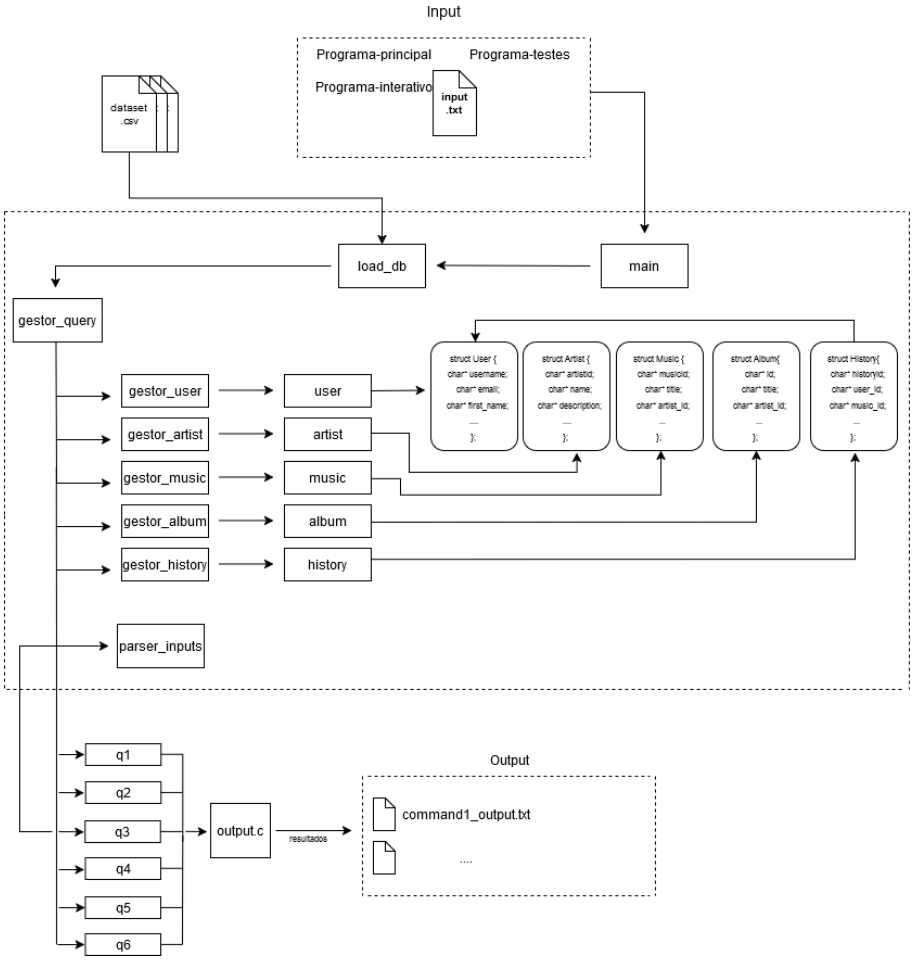
Por fim, documentámos detalhadamente o projeto e procurámos seguir sempre as melhores práticas de programação, com o objetivo de garantir um código claro, organizado e acessível.

2.1. Ajustes Realizados

A maioria dos ajustes realizados desde a primeira fase concentrou-se na modularidade e no encapsulamento. Após a identificação de algumas quebras de encapsulamento pela equipa docente, a nossa prioridade foi corrigir essas falhas, o que facilitou significativamente o desenvolvimento subsequente do projeto.

Procedemos ainda à divisão dos ficheiros em módulos distintos, garantindo uma melhor organização e identificação das diferentes partes do código. Consideramos que essas mudanças foram fundamentais e fizeram toda a diferença para o sucesso deste projeto, contribuindo para uma estrutura mais limpa, modular e eficiente.

2.2. Arquitetura do Projeto



2.2.1. Estruturas de Dados

Utilizamos diversas estruturas de dados ao longo da resolução das queries, escolhendo as mais adequadas para cada caso. Optamos por Hash Tables para todas as queries, devido à sua simplicidade e eficiência. Na query 4, complementamos com o uso de listas ligadas para armazenar os 10 artistas mais ouvidos, e uma árvore binária de procura, pois precisávamos agrupar os artistas por semanas dentro de um intervalo de datas fornecido no input.

2.2.2. Modularidade e Encapsulamento

O encapsulamento foi um dos principais pilares deste projeto, exigindo especial atenção desde as fases iniciais. Logo no início desta etapa, procedemos à correção de todos os erros identificados previamente, assegurando que as estruturas pertencentes aos diferentes módulos e contextos estavam devidamente encapsuladas e isoladas.

Para garantir um bom encapsulamento, implementámos cópias de memória sempre que foi necessário obter ou manipular dados, evitando acessos diretos aos blocos de memória originais das estruturas. Adicionalmente, utilizámos os padrões recomendados de *getters* e *setters* para controlar o acesso e a modificação dos dados, respeitando este princípio em todas as estruturas do projeto.

Em relação à modularidade, tomamos a decisão de adotá-la desde o início, deixando assim o projeto mais organizado e de fácil manutenção.

2.3. Queries

Todas as *queries*(tanto novas como antigas) precisaram de seguir um novo formato de output caso o input tivesse o *char* 'S'.

2.3.1. Query 1

Esta *query* sofreu pequenas alterações para passar a suportar artistas. Para além do nome, tipo e país, também foi necessário calcular o número de álbuns em que o artista surgia como autor individual e a sua receita total.

Foi utilizado o *artist_id* como chave para aceder à informação na *HashTable* e guardar a informação dos quatro primeiros campos (nome, tipo, país e número de álbuns do artista). No último campo, a receita total é calculada individualmente, isto é, acedendo diretamente ao artista na *HashTable* e indo acumulando a sua receita por reprodução. Quando um artista é do tipo *group*, a sua receita é partilhada por cada artista na lista dos id*_constituent,* sendo também necessário ir a cada artista individual e somar o resultado da receita por reprodução do artista principal (grupo) a dividir pelo número total de constituintes.

2.3.2. Query 2

Apesar da criação desta *query* ter sido feita na 1º fase do projeto. O aparecimento de um novo campo na entidade das músicas levou a que tivesse de ser feita uma validação extra para que a *query* voltasse ao seu normal funcionamento.

2.3.3 Query 3

Esta query foi criada na fase 1º do projeto não sendo preciso nenhuma alteração para esta funcionar.

2.3.4. Query 4

O objetivo desta *query* é identificar o artista que apareceu mais vezes no top 10 semanal dentro de um intervalo de tempo específico e devolver o seu id, tipo e a quantidade de vezes que esteve presente nesse ranking. Para resolver esta *query*, utilizámos três estruturas de dados.

Iniciámos com uma árvore binária, onde cada *node* representa uma semana do intervalo fornecido. Dentro de cada *node*, usamos *HashTables*, sendo a chave o id do artista e o valor o tempo que o artista foi ouvido na semana. A seguir, criamos uma *GList* que armazena a estrutura que contem o id do artista e o tempo de audição, a qual ordenamos em ordem decrescente e mantemos apenas os 10 primeiros artistas.

Após isso, transferimos os 10 artistas de todas as semanas para outra *HashTable*, onde a chave é o ID do artista e o valor é a quantidade de vezes que ele aparece no top 10 ao longo de todas as semanas. Finalmente, identificamos o artista com o maior número de aparições e devolvemos as informações solicitadas

2.3.5. Query 5

Esta *query* tinha de criar uma função que recomendava utilizadores com base nos seus gostos musicais e guardar num ficheiro a quantidade de utilizadores pedido nos inputs.

Para criar esta *query* optamos por utilizar o recomendador fornecido pelos docentes que pedia para fornecermos o id do utilizador alvo, uma matriz com os gostos por géneros de cada utilizador, uma matriz com o id de cada utilizador com a mesma ordem da matriz de gostos, um *array* com a ordem de géneros, o número de utilizadores, o número de genros e número de recomendações que a função deve devolver. Muitos destes dados eram bastante acessíveis, tirando tanto a matriz de id de utilizador com a matriz de pôr utilizador.

Para estas foi necessário criar funções de criação das matrizes. Esta percorria a *HashTable* dos utilizadores e para cada um destes adicionava o seu id a matriz e de id de utilizadores e ao mesmo tempo percorre o histórico associado a este utilizador e somava ao campo de géneros consoante o género da música atual.

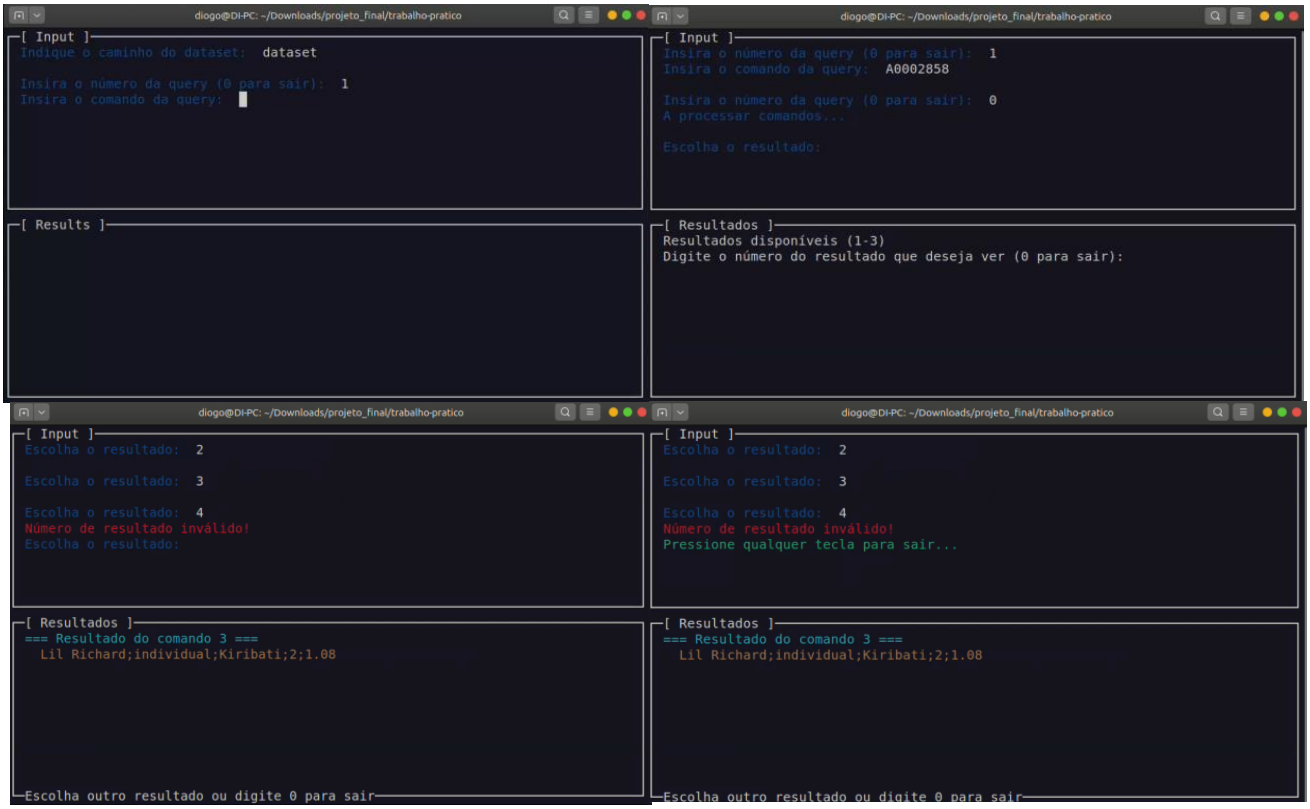
Por final manda cada um dos argumentos para o recomendador recebendo o uma matriz com os utilizadores recomendados guardando estes no ficheiro de output.

2.3.6. Query 6

Com o objetivo de devolver o resumo anual de um determinado utilizador, foram criados vários *structs* onde a informação relativamente a cada utilizador é guardada. O *struct* Historico é criado para cada utilizador e armazena o *musicid*, *timestamp* e *duration* em cada um para posteriormente serem usados para calcular o resumo anual.

Dentro do ficheiro q6.c estão definidas praticamente todas as funções usadas para esta *query*, sendo usados diferentes *structs* para cada função. De um modo geral, o processo é sempre o mesmo: percorrer o Historico em cada user, verificar o ano e devolver a informação pretendida.

2.4. Modo Interativo



2.5. Testes Funcionais

Desenvolvemos um módulo de testes para garantir a correta implementação das *queries* e avaliar o desempenho do nosso programa. Neste módulo, são analisados o tempo de execução de cada *query*, o tempo total de processamento, a validação dos resultados e o pico de memória utilizada pelo programa.

2.6 Desempenho

Nesta secção, apresentamos a análise dos desempenhos obtidos pelo computador de cada integrante do grupo durante o modo de testes:

Computador	**1**	**2**	**3**
Sistema Operativo	Ubuntu	Mint	Ubuntu
CPU	Intel® Core™ i5-1235U	AMD Ryzen 7 5800H	Intel® Core™ i5-6300
RAM	16 GB	16 GB	16 GB
Disco	512 GB SSD M.2	1024 GB SSD M.2	256 GB SSD M.2
Dataset Pequeno			
Query 1	7.512 s	7.718 s	10.062 s
Query 2	8.129 s	8.568 s	10.671 s
Query 3	7.923 s	8.037 s	9.757 s
Query 4	8.768 s	8.997 s	11.373 s
Query 5	8.453 s	8.522 s	10.646 s
Query 6	8.081 s	8.137 s	10.049 s
Tempo Total	48.866 s	49.982 s	62.560 s
Pico de memória	1260.39 MB	1260.27 MB	1260.58 MB
Dataset grande			
Query 1	57.213 s	57.319 s	71.268 s
Query 2	117.988 s	118.044 s	156.905 s
Query 3	56.754 s	56.861 s	70.932 s
Query 4	108.339 s	108.423 s	120.510 s
Query 5	66.023 s	66.118 s	75.604 s
Query 6	60.241 s	60.370 s	78.999 s
Tempo Total	466.558 s	467.139s	574.221 s
Pico de memória	2531.95 MB	2531.82 MB	2532.31 MB

2.7. Dificuldades sentidas

Durante o desenvolvimento deste projeto, a definição da lógica para a implementação das queries foi uma das maiores dificuldades que enfrentamos, especialmente pelo nosso compromisso em garantir a máxima eficiência.

Também encontramos desafios na otimização do projeto, já que procuramos constantemente um desempenho rápido e eficaz. No entanto, acreditamos ter superado essas dificuldades com sucesso.

3. Conclusão

Para concluir, acreditamos que abordámos este projeto da melhor forma possível, tendo sempre como prioridade a eficiência e o encapsulamento. Ao longo do desenvolvimento, percebemos o quão fundamental é a lógica de programação para alcançar um código bem otimizado, bem como a relevância da modularidade e do encapsulamento. Este projeto reforçou a importância de priorizar esses conceitos desde as etapas iniciais e levá-las adiante até ao final.

Algo que consideramos também indispensável, foi utilizar a ferramenta *Valgrind* para evitar a presença de *leaks*.

Consideramos que evoluímos significativamente ao longo da sua realização, especialmente graças às dificuldades que enfrentámos e que nos desafiaram a encontrar soluções criativas e eficientes.