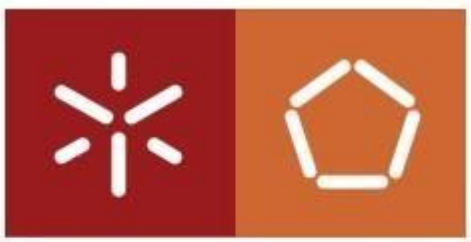
Projeto Laboratórios de Informática III

Fase II

Projeto desenvolvido por:
Pedro Teixeira(A103998), Rafael Fernandes(a104168), Diogo Fernandes(a104260)

Grupo 54

Licenciatura em Engenharia Informática



Universidade do Minho
Escola de Engenharia

Departamento de Informática
Universidade do Minho

Conteúdo

Capítulo 1
Introdução

No âmbito da unidade curricular de Laboratórios de Informática III, do ano letivo 2024/2025, foi-nos proposto o desenvolvimento de um projeto que consistia na implementação de uma base de dados em memória capaz de armazenar e processar dados fornecidos pelos docentes. Este projeto teve como objetivo principal introduzir os alunos aos princípios fundamentais da Engenharia de Software e da linguagem C, nomeadamente modularidade e encapsulamento, utilização de estruturas dinâmicas de dados, validação funcional, e medição de desempenho (tempo de execução, consumo de memória, etc.). Além disso, explorámos conceitos como compilação, linkagem, definição de objetivos de projeto com base nas suas dependências, depuração de erros, avaliação de desempenho e consumo de recursos, bem como gestão de repositórios colaborativos.
O projeto foi dividido em duas fases. Na segunda fase, o foco principal foi expandir as funcionalidades implementadas na fase anterior, garantindo a escalabilidade da solução para lidar com conjuntos de dados de maior dimensão e complexidade. Para tal, foi introduzido o processamento de dois novos ficheiros: o ficheiro de álbuns e o ficheiro de histórico. Esta fase incluiu também a criação de um modo interativo, que consiste numa interface no terminal para execução dinâmica do programa. O modo interativo permite ao utilizador fornecer a localização dos ficheiros de dados e realizar *queries* diretamente, proporcionando uma experiência mais intuitiva e versátil.
Nesta fase, priorizámos a aplicação rigorosa dos princípios de encapsulamento e modularidade, assegurando que o projeto apresentava uma estrutura robusta, bem organizada e facilmente extensível.

Capítulo 2
Desenvolvimento

Na primeira fase do projeto, a estrutura base foi desenhada com o objetivo de garantir uma organização clara e eficiente dos dados. Para tal, optámos por armazenar os dados lidos pelo *parser* em quatro tabelas *hash* distintas, cada uma correspondendo a um dos ficheiros de entrada fornecidos (exceto o histórico). Essa abordagem permitiu uma rápida indexação e acesso às informações, utilizando chaves únicas para cada entidade. Além disso, desenvolvemos estruturas auxiliares destinadas a armazenar estatísticas relevantes para as *queries*, calculadas durante o processo de *parsing*. Estas estruturas permitiam que as *queries* acessassem os dados processados de forma otimizada, reduzindo a necessidade de cálculos redundantes e melhorando o desempenho geral.

O fluxo geral das *queries* baseava-se na recolha e tratamento dos dados necessários, seguidos pela utilização de um módulo de output que escrevia os resultados num ficheiro de saída. Este design modular facilitou a separação de responsabilidades, permitindo que cada componente do sistema se focasse em funções específicas e evitando a sobrecarga de responsabilidades em módulos individuais.

Com a receção do enunciado da segunda fase, foi necessário ajustar e expandir a arquitetura para acomodar as novas funcionalidades e os conjuntos de dados adicionais. Entre as principais alterações e melhorias destacam-se:

- **Integração do histórico e dos álbuns:** Foi necessário adaptar as estruturas de armazenamento e criar métodos para processar e relacionar os novos dados.
- **Modo interativo:** A adição de uma interface no terminal permitiu que os utilizadores fornecessem o caminho para os ficheiros de dados e executassem *queries* de forma interativa, exigindo alterações na forma como os dados são carregados e processados.
- **Refinamento da modularidade e encapsulamento:** Foram revistas todas as funções e módulos do sistema para assegurar que seguiam os princípios fundamentais da Engenharia de Software. As tabelas *hash*, as estruturas de dados dinâmicas e as funções de manipulação de dados foram encapsuladas dentro de interfaces bem definidas, reduzindo a exposição direta das implementações.

Este foco na modularidade e encapsulamento visava garantir não apenas a robustez e a escalabilidade do sistema, mas também a sua manutenção. As decisões arquiteturais tomadas durante esta fase procuraram acomodar os requisitos da segunda fase, mantendo a clareza

no design do sistema e assegurando a continuidade do desempenho e da eficiência operacional.

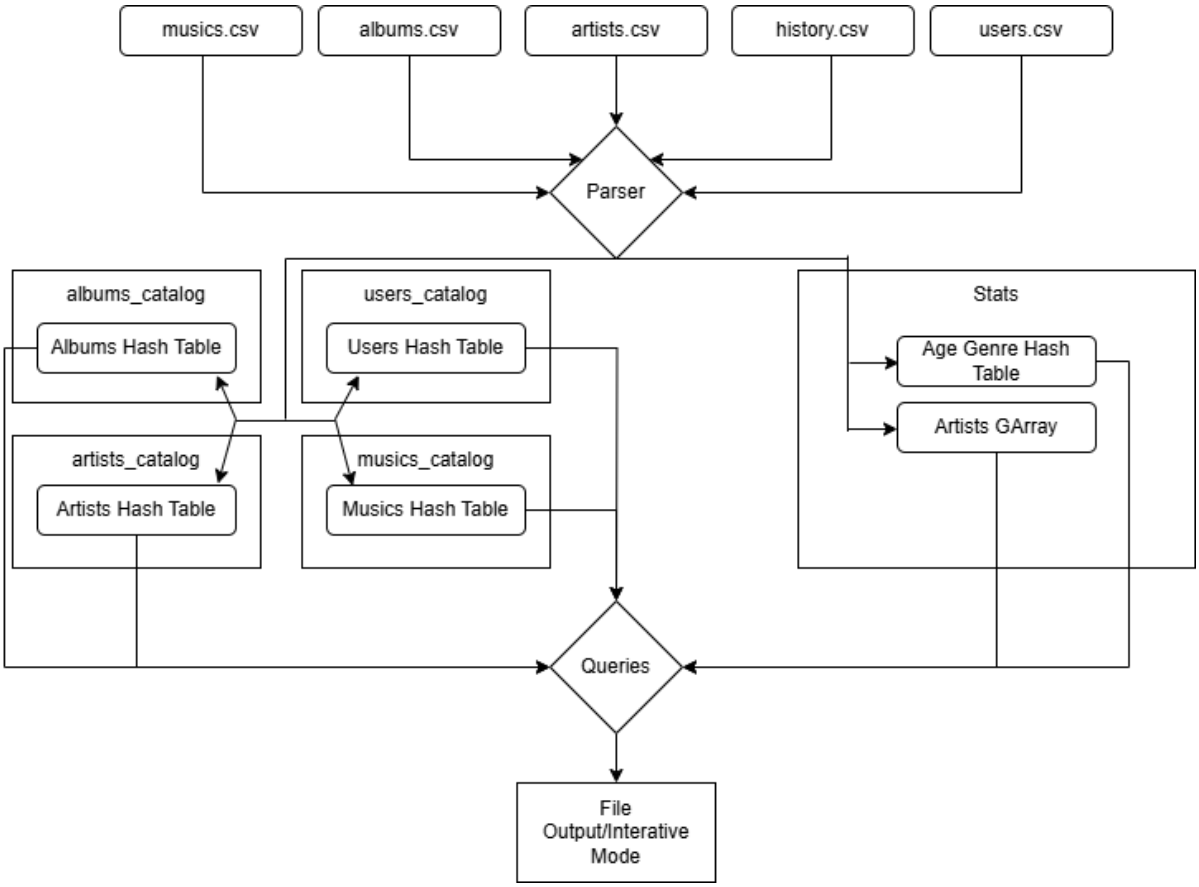


Figura 1: Diagrama da estrutura utilizada para a 2ª fase

QUERIES

Nesta fase optamos por apenas melhorar e atualizar as primeiras 3 *queries*.

Query 1:

A *query* 1 consiste em listar o resumo de um utilizador ou artista, consoante o identificador recebido por argumento. A *query* recebe como argumento o identificador único do utilizador ou do artista.

Posto isto, esta *query* foi a mais simples devido ao facto deste identificador único ser a chave de acesso a cada utilizador/artista inserido na *hash table*, logo, foi possível usar a função *lookup*(função predefinida da glib) para facilmente aceder ao utilizador/artista e às suas informações. Toda a informação impressa já estava armazenada na *hash table* o que mais uma vez contribuiu para a facilidade na resolução desta *query*.

Query 2:

A *query 2* consiste em listar os top N artistas com maior discografia. A query recebe como argumento o número de artistas que devem constar do output, podendo ainda receber (ou não) o filtro de país, sendo que quando presente só devem ser considerados artistas desse país.

Com vista a facilitar a execução desta query, foi criado um atributo "*Discography Time"* na entidade artistas e utilizada a componente *stats,* onde ao longo do *parsing* foram armazenados num *GArray* os artistas. Isto ajudou muito no processo da *query 2* devido à facilidade de organizar um *GArray* tendo em conta as funções predefinidas da *glib.*

Query 3:

A *query 3* consiste em listar os géneros de música mais populares numa determinada faixa etária. A query recebe como argumento as idades mínima e máxima da faixa etária.

Para facilitar a execução desta *query* foi utilizada a componente *stats,* onde ao longo do *parsing* dos users a *hash table* criada para este fim, onde para cada género(*key*) existe uma *hash table* cuja *key* é a idade dos *users* ao qual estão associados *likes que* são atualizados. Já dentro da *query* é utilizada a entidade "Género Musical" para simplificar a associação do total dos *likes* entre as diversas idades aos géneros. Posto isto, as diferentes entidades "GéneroMusical" são adicionadas a um *GArray* para, mais uma vez, organizar de forma mais efetiva o array usando as funções predefinidas da *glib*.

Capítulo 4
Otimizações

Com o objetivo de melhorar o desempenho e reduzir o consumo de memória do sistema, foram implementadas diversas otimizações ao longo do desenvolvimento da segunda fase do projeto. Estas otimizações abrangeram principalmente a reformulação das estruturas de dados e a reorganização de componentes críticos do sistema, de modo a garantir maior eficiência tanto na execução quanto na utilização de recursos. Abaixo destacam-se as principais melhorias implementadas:

1. Substituição de char* por int em Identificadores

Uma das maiores fontes de otimização foi a substituição de identificadores que anteriormente eram armazenados como strings (char*) por valores inteiros (int). Por exemplo:

- IDs de utilizadores, músicas, artistas e álbuns, antes representados por strings no formato "U0164294", "S0078723", foram convertidos para inteiros como 164294 e 78723.
- Esta alteração reduziu significativamente o consumo de memória associado a estas entidades, visto que um int ocupa menos espaço que uma string equivalente.
- Além disso, operações de comparação e indexação, como buscas e ordenações, tornaram-se mais rápidas, devido à simplicidade do processamento de inteiros em comparação a strings.

2. Conversão de Listas de Strings para Arrays de Inteiros

Os campos que armazenavam múltiplos identificadores, como os IDs de artistas associados a um álbum ou os IDs de músicas favoritas de um utilizador, foram convertidos de strings para arrays dinâmicos de inteiros (GArray):

- Antes: "['A0012345', 'A0067890']"
- Depois: GArray com valores [12345, 67890].

Esta mudança:

- **Eliminou a necessidade de operações frequentes de parsing**, como a divisão de strings por delimitadores, melhorando o desempenho em operações repetitivas.
- **Simplificou a manipulação interna dos dados**, permitindo operações mais diretas, como buscas e iterações nos IDs associados.

3. Alteração no Armazenamento de Durações

As durações de músicas, antes armazenadas como strings no formato "03:45", foram convertidas para valores inteiros representando o número total de segundos. Por exemplo:

- "03:45" passou a ser representado como 225.

Essa alteração:

- Reduziu o custo de operações matemáticas em durações, como somas e comparações.
- Simplificou a lógica interna do programa ao eliminar a necessidade de converter strings para inteiros sempre que as durações eram manipuladas.

4. Compactação de Tipos e Redução de Campos Não Necessários

Alguns campos redundantes ou pouco utilizados foram removidos, e tipos de dados mais compactos foram escolhidos sempre que possível:

- O campo recipe_per_stream, anteriormente uma string como "0.02", foi convertido diretamente para um valor double, reduzindo o espaço ocupado e permitindo cálculos diretos.
- Tipos como char* foram substituídos por GArray<int> em listas de IDs, reduzindo a fragmentação de memória causada por múltiplos alocadores dinâmicos.

5. Uso Eficiente de Estruturas da GLib

Aproveitamos estruturas de dados fornecidas pela **GLib**, como:

- GArray para listas dinâmicas de inteiros, substituindo arrays estáticos ou listas de strings.
- GHashTable para mapeamento eficiente de dados, garantindo acessos rápidos e gerenciamento de memória integrado.

6. Otimizações no Parsing

Durante o processo de parsing:

- Os dados foram processados de forma incremental, e as estatísticas necessárias para queries foram calculadas e armazenadas diretamente nas estruturas relevantes, reduzindo a necessidade de cálculos posteriores.
- Utilizámos estruturas auxiliares para evitar duplicação de cálculos e acessos desnecessários aos dados originais.

7. Gestão de Memória

Para evitar *memory leaks*, foram adotadas boas práticas de libertação de recursos:

- Cada função que alocava memória era acompanhada por uma função correspondente para libertar esses recursos.
- Estruturas compostas, como tabelas hash e arrays dinâmicos, foram destruídas utilizando métodos apropriados da **GLib**.

Impacto das Otimizações

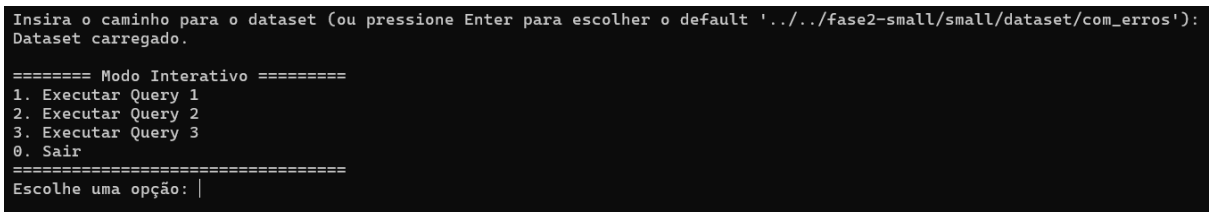
Estas otimizações permitiram:

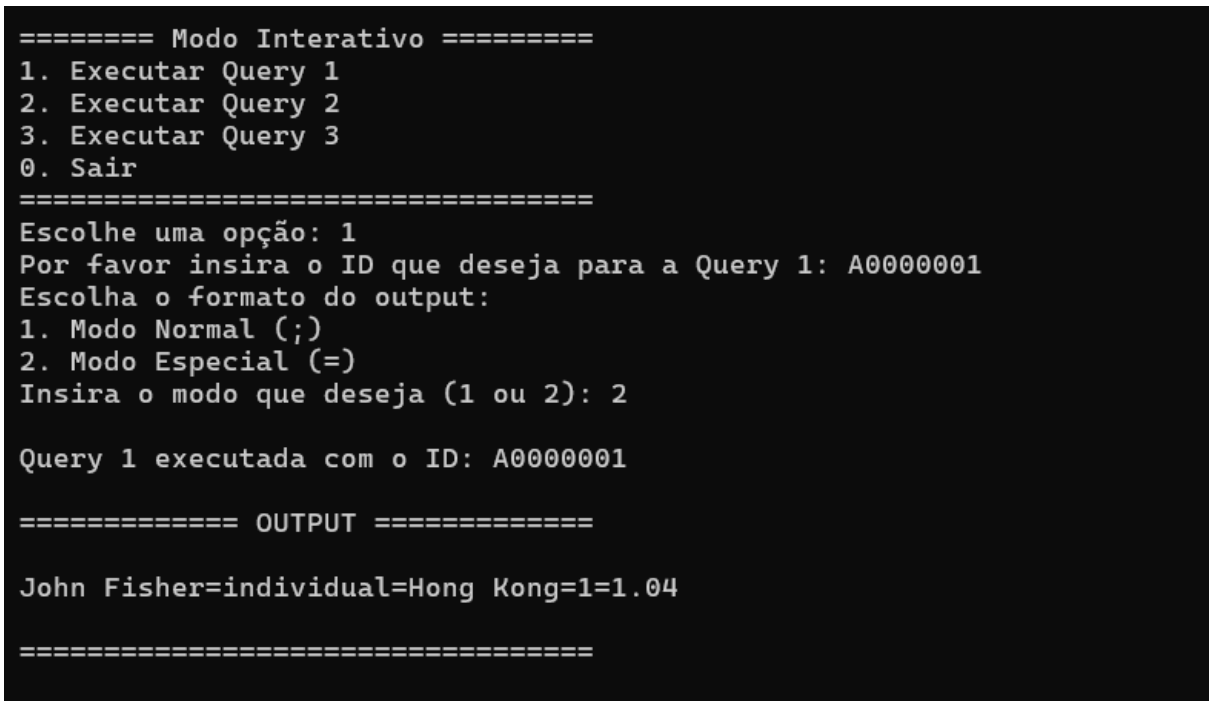
- **Redução significativa no uso de memória**, essencial para lidar com o aumento do tamanho do dataset.
- **Melhoria no desempenho**, com tempos de execução menores, especialmente em operações repetitivas e consultas envolvendo grandes volumes de dados.
- **Escalabilidade do sistema**, garantindo que as novas funcionalidades pudessem ser integradas sem comprometer o desempenho ou a estabilidade.

Estas melhorias refletem um esforço consciente para alinhar o projeto aos princípios de eficiência e modularidade, pilares fundamentais do desenvolvimento de software de qualidade.

Capítulo 5
Modo Interativo

O modo interativo desenvolvido neste projeto tem como objetivo fornecer ao utilizador uma interface intuitiva e funcional para explorar os dados armazenados em memória e realizar consultas dinâmicas. Este modo foi projetado para ser executado diretamente no terminal, permitindo ao utilizador carregar os ficheiros de dados, realizar *queries* de forma interativa, e obter resultados em tempo real. A sua implementação reflete os princípios de modularidade e encapsulamento, garantindo uma arquitetura robusta e escalável.





Capítulo 6
Desempenho e Testes

Este capítulo detalha o processo de teste realizado no projeto, focando nos conjuntos de dados. Os testes desempenham um papel crucial no desenvolvimento de software, garantindo a qualidade e confiabilidade do sistema. Neste contexto, realizamos testes abrangentes nos conjuntos de dados para validar o desempenho, a precisão e a escalabilidade.

ASUS Vivobook 16	Tempo de execução	Memória gasta
Teste 1 (Regular Dataset)	11.86 seg	158428 KB
Teste 2 (Large Dataset)	69.01 seg	467976 KB

Capítulo 7
Conclusão

Este projeto representou, para nós, um grande desafio, especialmente durante esta segunda fase. A complexidade aumentou consideravelmente devido à introdução de novos conceitos, funcionalidades mais avançadas e requisitos rigorosos de desempenho e modularidade. Estas exigências testaram não apenas as nossas habilidades técnicas, mas também a nossa capacidade de organização e gestão de tempo.

Uma das principais dificuldades enfrentadas foi conciliar o desenvolvimento deste projeto com outras responsabilidades académicas e atividades paralelas, o que resultou em uma gestão de tempo mais apertada e, em alguns momentos, insuficiente. Contudo, encaramos este obstáculo como uma oportunidade para melhorar a nossa resiliência e capacidade de priorizar tarefas, competências essenciais no contexto profissional.

Apesar das dificuldades, sentimos que conseguimos evoluir significativamente em relação à fase anterior. Implementámos melhorias fundamentais na modularidade e encapsulamento do código, reduzindo redundâncias e otimizando o desempenho, ao mesmo tempo que tornámos o sistema mais escalável e preparado para lidar com grandes volumes de dados. Além disso, a criação de um modo interativo permitiu-nos explorar conceitos como interação com o utilizador e gestão dinâmica de dados, ampliando o escopo do nosso aprendizado.

No final, este projeto foi uma experiência rica e desafiadora que nos permitiu aplicar e consolidar os conhecimentos adquiridos ao longo do curso, preparar-nos para futuros desafios e adquirir uma visão mais prática sobre os princípios fundamentais da Engenharia de Software e da linguagem C.