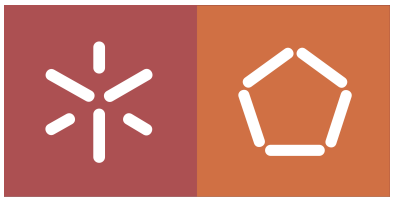

Universidade do Minho
Escola de Engenharia

Licenciatura em Engenharia Informática

Grupo 11

Tiago Matos Guedes A97369
Diogo Filipe Durães Ribeiro A89981
Luís Filipe Araújo Ferreira A98286

Laboratórios de Informática III

2º Ano, 1º Semestre

Departamento de Informática

07 de janeiro de 2025

Índice

1 Introdução

Dando seguimento ao trabalho realizado para a 1ª fase do projeto, proposto pela unidade curricular Laboratórios de Informática III, nesta 2ª fase surgiram novos desafios, desde o tratamento e raciocínio sobre dados mais complexos, à implementação de um programa interativo com o utilizador. Apresentamos de seguida a nossa aproximação a esses novos desafios, descrevendo essa aproximação e enunciando as dificuldades encontradas pelo caminho.

2 Modelação

2.1 Arquitetura de Aplicação

A arquitetura do projeto é constituída por 5 partes principais:

- **Entidades**: Contendo os objetos a serem tratados no projeto e distintos entre si. Relacionados com os dados já tratados temos, Álbum, Artista, Histórico, Músicas e Utilizadores, relacionando alguns desses dados entre si temos, Discografia e Resumo (Wrapped).
- **Estatísticas**: Constituída por entidades auxiliares que vizam optimizar o trabalho realizado pelas queries através de pré-calculo no parsing.
- **Gestores**: Estabelecem a conexão entre as entidades relacionadas aos dados recebidos como input do utilizador, e as suas respetivas estruturas de dados. Controlam essas estruturas de dados, tendo capacidades de inserção remoção entre outras manipulações.
- **Parser**: Tem como função ler e manipular os dados, sejam dos ficheiros de dados, sejam do ficheiro de queries a executar sobre esses mesmos dados, de modo a serem inseridos nas estruturas de dados, as hashTables de cada entidade. No caso das queries, prepara os dados, trata de subestruturas necessárias para a realização das queries e decide quando e qual query a executar.
- **Queries**: Onde se encontra exclusivamente o código para a resolução das 6 queries propostas.
- **Utils**: Conjunto de ferramentas disponíveis a todo o projeto.
- **Writter**: Utilizado após a realização da lógica das queries, responsável pela formatação e respetivo output das mesmas.

Figura 1: Maquete do Projeto

2.2 Organização dos Dados

2.2.1 Estruturas de Dados

Nos gestores as estruturas de dados utilizadas são: *Hashtable* de Artistas, para as músicas uma *Hashtable* de Músicas, e uma estrutura *Wrapped* baseada na estrutura de músicas, para os utilizadores uma *Hashtable* de Utilizadores, uma *Hashtable* e um *GPtrArray* baseados nos dados dos utilizadores, uma *Hashtable* de Álbuns, uma *Hashtable* de Históricos, duas *Hashtable* baseadas nos dados dos históricos e dois *GPtrArray* também relacionados com os históricos. Foi ainda criada uma estrutura auxiliar *Discography* que será aprofundada juntamente com outras estruturas a seguir. Para ajudar à implementação de algumas destas estruturas recorremos à biblioteca *GLib-2.0* para facilitar o uso e minimizar erros.

- **Wrapped** A acompanhar a *Hashtable* das músicas, a estrutura é constituída por varíaveis relevantes à resolução da query 6 proposta.
- **Discography** Estrutura essencial para a resolução da query 2 proposta. Com varíaveis para armazenar os dados relevantes para a sua resolução e ordenada por ordem decrescente, otimizando o output.

2.2.2 Porquê estas Estruturas?

A escolha das *Hashtables* e dos *GPtrArrays* visou otimizar o tempo de execução em cenários com grandes quantidades de dados visto terem funções definidas na biblioteca *GLib-2.0* que as tornam eficientes para a função. Para a realização da 4ª Query, foram utilizadas tabelas para a eficiência da inserção de elementos e atualização dos mesmos. Para a resolução da 5ª Query foram utilizadas tabelas novamente com o mesmo motivo da 4ª Query. Para a resolução da 6ª Query, optamos por criar uma estrutura auxiliar *Wrapped* para nos acompanhar durante a resolução do query, fornecendo a informação necessária para o output de maneira direta, sem efetuar procuras ou cálculos acrescidos.

3 Encapsulamento

Como compreendido no relatório, o grupo levou sempre como referência a importância da modularidade no contexto do projeto, e por isso implementámos as seguintes estratégias ao longo do trabalho:

- Utilização de estruturas opacas: definição das estruturas ao longo do projeto foi apenas escrita em ficheiros *.c* e as suas declarações nos respetivos *.h*, limitando o acesso ás estruturas para apenas às enunciações do módulo localizadas no ficheiro *.h* do mesmo.
- Criação de métodos getters e setters dentro de cada módulo que achámos relevante para manter uma camada de proteção de acesso às estruturas e mantê-las assim opacas.

4 Implementação

4.1 Verificações

Nesta 2ª fase do projecto, mantivemos a sintaxe de verificação dos ficheiros e aplicamos aos novos, albuns e históricos. Optamos desta vez por fazer a validação de de cada ficheiro num módulo respetivo a cada um.

4.2 Query 1

4.2.1 Método adotado

Para a realização da 1ª *query*, que tem como objetivo obter informações sobre um utilizador ou artista desejado, consoante o identificador recebido por argumento, a função query1 tem como função procurar, calcular e retornar os dados principais do utilizador ou artista através do seu username ou no caso do artista o seu ID. A função faz um pré-cálculo da informação necessária á medida que faz parse dos históricos, popula duas tabelas no gestor de artistas denominadas *"num_albuns_table"* e *"rep_musics_table"*, que guardam o número

de albúns ao qual aquele artista pertence e as repetições das músicas do artista respetivamente.

Deste modo a query 1 apenas necessita de consultar a informação que necessita sendo um utilizador ou um artista, utilizador apenas informações de entidade, no caso do artista o número de albúns e a receita gerada. Para o cálculo da receita gerada, é efetuado dentro da query 1 através de qual tipo for o artista. Se for do tipo individual gera a lista de quais coletivos esse artista individual faz parte e faz o cálculo através da fórmula do cálculo individual, sendo adicionada a sua parte em cada coletivo que fizer parte. Por ventura, se for do tipo grupo apenas multiplica as repetições pela receita.

4.3 Query 3

4.3.1 Método adotado

Tendo sido atualizada a 3ª Query, achamos pertinente explicar as mudanças e optimizações desta query, passando a receber um gestor de utilizadores que contém encapsulado um GPtrArray com dados já pré-calculados.

4.3.2 Funcionamento e Aplicação

Os dados presentes no GPtrArray *"user_likes_array"* foram pré-calculados, com auxilio de uma HashTable *"user_likes_table"* que tem como chave uma idade de um utilizador e como valor uma estrutura que encapsula as informações relevantes do utilizador, esta estrutura contém um array de géneros e um array de gostos. Sempre que um utilizador novo com uma idade já existente na tabela, o processo de inserção iria verificar se já existia um elemento com a mesma idade, nesse caso, iria atualizar os géneros e os gostos que cada género tem, com a nova informação desse utilizador.

Deste modo, evitamos uma quantidade considerável de elementos por *username* e passamos a ter dados reduzidos para idades, contendo para cada idade todos os géneros e gostos já pré-calculados.

Sempre que a query é chamada, só é necessário percorrer o array *"user_likes_table"* e para cada elemento do array verificar se está compreendido entre o intervalo de idades fornecido, populando um array de géneros e de likes, inicializado na query que irá representar o somatório de likes para cada género.

No fim, é feita uma ordenação dos géneros com mais likes e é escrito o resultado final no ficheiro.

4.4 Query 4

4.4.1 Método adotado

Para a realização da query 4, foi feita a seguinte análise, povoamento de uma tabela *"week_artist _duration_table"* que contém para cada semana um valor que corresponde a uma tabela de artistas *"artist_ht"*, esta tabela contida, possui chave que representa o id do artista, e como valor um artista associado. Mais tarde, é povoada uma tabela *"week_top10_table"* que contém para cada semana uma lista de top 10 artistas já ordenados tendo em conta os critérios definidos no enunciado. Por fim, é povoada uma ultima tabela *"artist_count_table"* que irá armazenar para cada artista o número de vezes que ele esteve no top 10.

Deste modo, a query 4 apenas necessita de verificar se o intervalo de datas fornecido corresponde a uma semana existente na tabela, se corresponder, é determinado o artista que esteve mais vezes através de funções auxiliares.

4.5 Query 5

4.5.1 Método adotado

Para a realização da query 5, foi pré-calculada uma tabela *"genres_listened _table"* e um array *"genres_listened_array"* que contém a informação necessária de cada utilizador candidato a ser semelhante a um dado utilizador.

Construíndo um array de generos e um array que representa para cada genero as vezes foi ouvido, através desta estrutura, podemos comparar os dados de um utilizador com um utilizador alvo, que procura recomendações.

No processo da query 5, é populado um array de utilizadores semelhantes *"similar_users_array"* seguindo os critérios de semelhança e é, ordenado do utilizador mais semelhante ao menos semelhante. Por fim, é feita uma iteração pelo *GPtrArray* para escrever as N recomendações.

4.6 Query 6

4.6.1 Método Adotado

É usada uma estrutura de dados auxiliar ao cálculo do resumo anual de um utilizador. *Wrapped* faz parte do gestor de músicas e é constituída por:

- **ano**: do tipo *char**, representa o ano do resumo a calcular, considerado por motivos de comparação.
- **userId**: do tipo *long int*, representa o identificador único do utilizador, considerado por motivos de comparação.
- **albuns**: do tipo *int***, duplo array destinado aos identificadores únicos do album e o seu tempo de audição, mesmo índice.
- **horas**: do tipo *int**, iniciado para 24 unidades, incrementado conforme necessidade em segundos, cada índice representa a (hora-1) do dia.

- **generos**: do tipo *int**, tendo interpretado haver apenas 10 géneros distintos, é iniciado para 10 unidades, sendo cada índice um género subentendido (ordem alfabética), é incrementado conforme necessidade em segundos.

Blues	Classical	Country	Electronic	Hip Hop	Jazz	Metal	Pop	Reggae	Rock
0	1	2	3	4	5	6	7	8	9

Tabela 1: Géneros por índices

- **dias**: do tipo *int***, duplo array, simulando um calendário. A alocação de memória é feita para 12 arrays, cada um com uma alocação de 31 unidades. Consideramos todos os meses com 31 dias por razões de simplificação. É incrementado em segundos na posição [mês][dia] respetivo ao timestamp.
- **artistsTimes**: do tipo *ArtistsTimes** que representa uma lista ligada, seguindo a estrutura:
  - **artistId**: do tipo *long int**, representa a lista de artistas de uma determinada música.
  - **listTime**: do tipo **int**, representando o tempo em segundos ouvido por um utilizador, de um artista ou artistas se coletivo. Incrementado em segundos.
  - **listMus**: do tipo **long int***, responsável por guardar identificadores únicos de músicas, para comparar e nunca adicionar músicas repetidas.
  - **next**: do tipo **ArtistsTimes***, um apontador para o resto da lista com dados para diferentes artistas

4.6.2 Funcionamento e Aplicação

Através da função *foreachHistory*, a query 6 percorre a *Hashtable* com recurso à função *yearResumed* à procura do *userId* já definido na estrutura *Wrapped*. Caso encontre verifica-se se o ano do *timeStamp* desse histórico é igual ao ano também definido na estrutura auxiliar.

Casos os dois casos se verifiquem, obtemos os dados da música, através do *musicId* presente no histórico que satisfez as condições, através de um *searchMusic* e as necessárias funções auxiliares para fazer uma cópia dos dados da música encontrada.

Após recolha dos dados relevantes para a query, através de funções auxiliares da estrutura *Wrapped*, preenchemos a estrutura, maioritariamente incrementando nos respetivos arrays os segundos da duração do histórico e preenchendo os arrays de *artistsTimes* caso ainda não existam.

Finalizada a função *foreachHistory*, o restante código na função *query6* (função chamada para resolver cada query 6), interpretando o pedido na linha da query, extrai os valores necessários e escreve no ficheiro de output.

5 Desempenho

- Dispositivo 1:
  - Processador: Intel Core i5-12450H 2.50GHz
  - Memória RAM: 16GB
  - SO: Windows WSL2 Ubuntu 22.04.6
- Dispositivo 2:
  - Processador: Intel Core i3-8130U 2.2GHz
  - Memória RAM: 8GB
  - SO: Windows WSL2 Ubuntu 22.04.6
- Dispositivo 3:
  - Processador: Intel Core i7-7700HQ 2.8GHz
  - Memória RAM: 16GB
  - SO: Windows WSL2 20.04.5 LTS

	Dispositivo 1	Dispositivo 2	Dispositivo 3
Query 1	0.002031	0.003817	0.013721
Query 2	0.000278	0.000628	0.003149
Query 3	0.000087	0.000160	0.001122
Query 4	0.003562	0.006795	0.014018
Query 5	0.703879	1.371434	1.829683
Query 6			3.645184
Tempo de Execução Total	3.1841s	6.0627s	6.4640s
Memória Utilizada	510MB	510MB	510MB

Tabela 2: Análise de Desempenho com Dataset simples

Pelos testes efetuados consegue-se observar um maior tempo de execução para a query 5 comparativamente com as restantes devido á utilização de diversas estruturas para o cálculo da mesma, sendo que nas restantes se utiliza um menor de fluxo de dados comparativamente á 5.

6 Dificuldades Sentidas

A resolução da query 4 apresenta resultados que não condizem com os esperados, em certos casos apresenta-se uma diferença de uma semana, em outros casos o *artist_id* não condiz com que foi calculado pelo nosso grupo, esta diferença deve-se possivelmente à incompreensão da query 4.

A resolução da query 5 apresenta resultados que também não condizem com os esperados, acreditamos que tal se deve ao critério que o grupo definou para determinar se um utilizador é candidato a ser semelhante e/ou também devido à maneira como é ordenada os nossos utilizadores semelhantes. O grupo não foi capaz de resolver o problema e esta query permaneceu com esse problema.

A resolução da query 6 não se demonstrou eficiente pois apesar de apresentar resultados na direção certa para serem todos corretos, quando os ficheiros de dados são de maior dimensão, essa ineficiência vem ao destaque pois o programa é terminado forçadamente por uso excessivo de recursos. Apesar da lógica do pensamento considerarmos estar correta entre muitas outras melhores, a sua execução ficou aquém.

Com a proposta de novas queries mais complexas, a exigência das mesmas obrigou o grupo a um trabalho redobrado com constantes imprevistos, consideramos as queries da fase 2 mais complicadas do que na fase anterior.

Foram encontradas algumas dificuldades ao longo do projeto na resolução de *memory leaks*, inicialmente não foi feito todos os *free's* e por consequência, foi necessária uma análise extensa com o auxílio do *valgrind* para resolver este problema.

7 Conclusão

Em suma, o grupo acredita que a maior parte dos resultados esperados com o projeto foram obtidos com sucesso. Nesta segunda fase o grupo apenas sentiu mais dificuldades com a realização das novas queries devido a serem uma grande mistura de dados para serem processados, fazendo com que apenas uma parte tenha sido atingida com sucesso na parte das queries. Foi implementado adicionalmente o menu interativo e melhorado o programa de testes mostrando agora melhor a informação obtida pelo mesmo e mais organizada.

Ao que toca à modularidade e ao encapsulamento consideramos que o grupo tenha cumprido os objetivos pretendidos pelo projeto, respeitando sempre a cada momento do mesmo todos os princípios.

Nesta UC foram aprendidos conceitos para uma melhor segurança, melhor organização, utilização eficiente de recursos e de tempo gasto pela máquina, reutilização de código, etc. pelo que esperamos certamente poder vir a utilizá-los e se possível consolidá-los ainda mais no futuro.