UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO E
ENGENHARIA DE COMPUTAÇÃO

GUILHERME KOHL, LUCAS SLONGO SCHIAVO

DESENVOLVIMENTO DE SOFTWARE INDEXADOR DE MÚSICAS

Relatório apresentado como requisito parcial para a obtenção de conceito na Disciplina de Classificação e Pesquisa de Dados.

Orientador: Prof. Dr. Leandro Krug Wives

Porto Alegre
2024

SUMÁRIO

1 INTRODUÇÃO

Este relatório tem o objetivo de descrever o desenvolvimento de uma aplicação cuja principal funcionalidade é a indexação de músicas. Esta é uma aplicação desktop executada localmente que permite o usuário obter informações sobre as músicas de suas *playlists*, possibilitando-o conhecer mais sobre seu gosto musical.

Para extrair as informações necessárias, foi usada a API do Spotify, maior serviço de *streaming* de músicas da atualidade. Além disso, implementou-se diferentes estruturas de arquivos a fim de possibilitar variadas funcionalidades de busca de dados. A aplicação foi desenvolvida integralmente com a linguagem de programação Python, que possui ótimo suporte de conexão com a API.

1.1 Descrição do problema

A maneira mais fácil de consumir música hoje é através de serviços de streaming. Neles, o usuário é capaz de salvar várias músicas em diferentes playlists, como também "favoritar" artistas e álbuns. A maior plataforma de *streaming* de músicas, o Spotify, possui um grande catálogo incluindo tanto grandes artistas internacionais como compositores independentes e de pouco renome. Por conta disso, é comum que um usuário salve em sua biblioteca milhares de músicas, perdendo, assim, o contato com os artistas, álbuns e músicas.

Pensando nisso, foi idealizado um *software* que possibilita a indexação de músicas a partir de seus diferentes atributos, como nome, duração e popularidade. Dessa forma, é possível conhecer mais sobre seu próprio gosto musical, reestabelecendo uma relação com a música que parece que foi deixada na época dos discos e CD's.

1.2 Funcionalidades previstas

Para atingir a melhor experiência ao usuário, as seguintes funcionalidades foram pensadas:

1. Indexar um arquivo de músicas a partir de diferentes atributos, sendo estes: nome, popularidade, duração

2. Calcular estatísticas sobre as músicas, como média de popularidade de todas as músicas
3. Exibir as informações através de uma interface no terminal, fazendo uso de tabelas paginadas

1.3 Possíveis estratégias para solução do problema

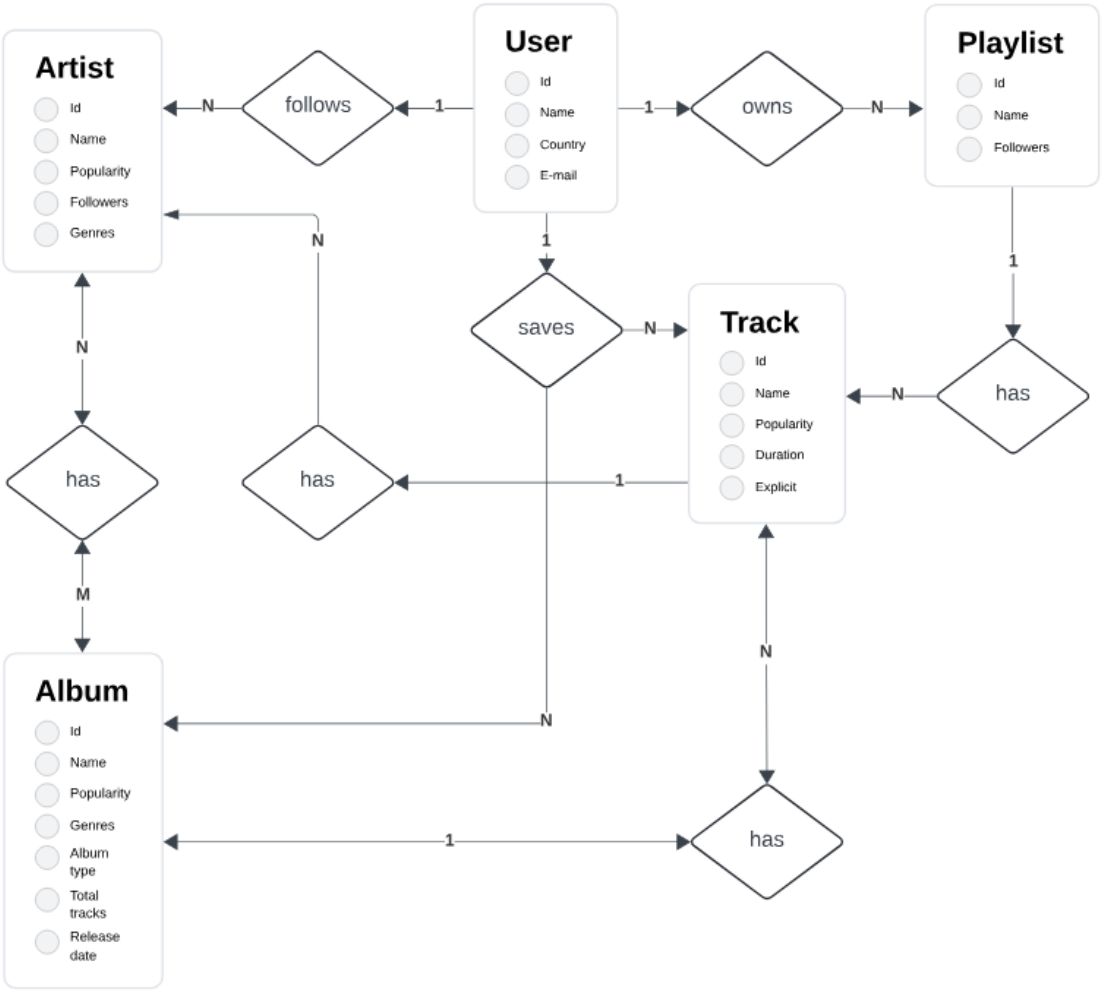
Para lidar com um grande volumes de dados, é necessário a utilização de arquivos em disco uma vez que não é possível armazenar tudo na memória principal. Nesse contexto, os mais adequados seriam os arquivos do tipo binário devido a sua ótima capacidade de compactar informação.

Apesar de eficientes para guardar informação, os arquivos apresentam desafios para recuperar estes dados. Para superá-los, pode-se usar tanto arquivos seriais (registros em ordem de inserção) ou sequenciais (registros ordenados por chave) juntamente com estruturas de índices de acesso (como árvores-B, TRIE e PATRICIA). Dessa forma, é possível encontrar os dados salvos sem precisar efetuar buscas pelo arquivo inteiro.

1.4 Projeto de arquivos

Antes de definir o modo como os arquivos seriam organizados, foi feito um planejamento das entidades (*structs*) que seriam utilizadas no programa e seus relacionamentos. Para melhor visualização, elaborou-se um diagrama ER (*entity relationship*), como mostra a figura:

Diagrama 1.4 – ER (entity relationship)



É importante reparar que certas relações não são mútuas. Através de uma *track* (música) é possível acessar seu álbum e seus compositores (artistas), no entanto, através de um artista não é possível acessar diretamente suas músicas. Nesse sentido, a direção da seta indica quem é o "executor" da relação indicada pelos losangos, por exemplo, o usuário é quem possui a *playlist*, enquanto tanto um álbum quando uma música "se possuem".

Após discussão sobre as diferentes formas de estruturar os arquivos, optou-se pelos arquivos seriais por conta de sua eficiência na inserção. Apesar de serem menos práticos na busca, tal dificuldade seria resolvida através das árvores de índice. Nesse sentido, a forma pensada para organizar os dados foi um arquivo principal e diferentes estruturas de índice, uma para cada atributo.

Em um primeiro momento, as árvores-B pareceram as mais apropriadas para a aplicação uma vez que são compactas e compatíveis com os diferentes atributos pensados. No

entanto, ao longo do desenvolvimento da aplicação as árvores TRIE e os arquivos invertidos foram, também, implementados.

2 DESENVOLVIMENTO

Após o devido planejamento acerca da estrutura de arquivos e relacionamento de entidades, o desenvolvimento do código foi iniciado. Para isso, utilizou-se o editor de texto Visual Studio Code e o manejador de repositórios Git Hub (o repositório dos códigos se encontra nesse link). A linguagem de programação escolhida foi Python devido ao seu ótimo suporte a API do Spotify.

2.1 Base de dados

Para desenvolver a aplicação, era necessário criar uma base de dados para testar o funcionamento das estruturas de índice. Os dados foram coletados através de API fazendo uso de um *script* em Python.

2.1.1 Spotify API

Para uso do API, é necessário ter uma conta no Spotify e criar uma nova aplicação. Com ela, se obtém um ID de usuário e uma chave de acesso. Com isso é possível solicitar um *token* que permite coletar os dados do sistema. O *token* fica válido por uma hora, depois é necessário solicitar outro. A maneira mais simples de resolver esse impasse é sempre solicitar um novo *token* no início de qualquer script que acesse a API.

A API dá acesso a informações públicas e privadas (apenas com o consentimento do usuário) sobre músicas, álbuns, artistas, *playlists* e usuários. Ela possui uma variedade de chamadas que possibilitam a extração dos mais diferentes dados. É possível acessar objetos concretos como também informações bem específicas, por exemplo, o quão barulhenta (em decibéis) uma música é.

2.1.2 Montagem da base

Basicamente, o que o *script* faz é, após obter autorização do usuário, carregar todas as suas *playlists* e percorrer música por música de cada uma, salvando em uma lista de *playlists*. É importante ressaltar que o que a API retorna é um JSON para cada música, e que este, além de não estar no formado ideal para utilização dentro do código, possui mais atributos dos que

os desejados. Para resolver este problema, o programa converte o JSON para um dicionário e este tem suas informações filtradas.

Por fim, com as músicas extraídas e preparadas só faltava salvá-las em binário. Dessa forma, concluiu-se a montagem do arquivo principal, com todos os registros que seriam utilizados ao longo da aplicação.

2.2 Estruturas

Ao longo do desenvolvimento do programa, foram criadas classes para os objetos extraídos da API juntamente com três estruturas de índice.

2.2.1 Classes

Principalmente, cinco classes foram implementadas:

1. Álbum: id, nome, popularidade, gêneros, tipo, nº de músicas, data de lançamento
2. Artista: id, nome, popularidade, nº de seguidores, gêneros
3. Playlist: id, nome, músicas
4. Músicas: id, nome do artista, nome, popularidade, duração, explicitude
5. Usuário: id, nome, país, e-mail

Nem todos os atributos são de fato utilizados ao longo da aplicação, mas podem ser úteis em modificações futuras.

2.2.2 Estruturas de índice

Em um primeiro momento, a ideia seria utilizar uma B-tree para cada atributo de música. No entanto, ao longo do desenvolvimento notou-se que esta talvez não fosse a mais adequada para alguns casos. Por isso, arquivos invertidos e árvores TRIE foram implementadas:

- B-tree: buscas por nome; cada elemento dentro do nodo é um par (nome, ponteiro para posição de arquivo)
- TRIE: buscas por prefixo; cada nodo possui um booleano indicando se este é um fim de palavra e seu ponteiro para posição de arquivo
- Arquivo invertido: classificação quanto à popularidade (atributo número de 0 a 100 frequentemente repetido entre músicas); é um dicionário cuja chave é a popularidade e seus valores são uma lista de ponteiros para posição de arquivo

Sempre que uma busca é feita, os ponteiros são usados para ler o registro correspondente a chave e, assim, obter a informação completa sobre aquele objeto.

2.3 Interface

Para melhor visualização das informações obtidas, foi implementada uma interface que funciona dentro do terminal. Esta permite o usuário interagir com o código solicitando diferentes informações e, quando conveniente, mostrar os resultados em tabelas paginadas. Cada linha da tabela é um registo de música completo.

Além de exibir os resultados, a interface permite a reordenação instantânea da tabela, invertendo sua ordem. Também é possível, após cada busca, efetuar novas com diferentes formas de ordenação.

2.4 Bibliotecas

Para desenvolver a aplicação, foram necessárias diferentes bibliotecas, incluindo algumas que não são padrão do Python:

- math: operações matemáticas mais elaboradas
- time: funções de *timeout*
- rich*: design de interface via terminal
- keyboard*: leitura do teclado
- datetime: formatação de tempo em horas:minutos:segundos
- pickle: serialização de objetos
- dotenv: manipulação de arquivos .env
- os: manipulação de demais arquivos no diretório
- spotipy*: execução de chamadas da API

As bibliotecas marcadas com asterisco não são padrão e necessitam ser instaladas, seja globalmente ou em ambiente virtual.

3 MANUAL DE USO

Essa aplicação é simples e por isso não necessita de extensos passos para configuração ou requisitos de sistema complexos. Além disso, possui uma interface clara que não deve apresentar dificuldades de uso ao usuário.

3.1 Requisitos

Para a aplicação funcionar adequadamente é necessário ter Python instalado no computador na verão correta. O código foi escrito com a versão 3.10.6, porém, qualquer versão a partir da 3.8 deve funcionar também.

Em se tratando das bibliotecas, é necessário instalar “rich” e “spotipy”. Elas podem ser facilmente instaladas globalmente com o pip em um terminal:

- Windows: py –m pip install “nome-da-biblioteca”
- Unix/macOS: python3 -m pip install “nome-da-biblioteca”

3.1.1 Virtual Enviroment

Caso desejado, é possível instalar as bibliotecas localmente em ambiente virtual. Para isso, é necessário certa configuração:

1. Vá no diretório do projeto e execute:
   Windows: py -m venv .venv
   Unix/macOS: python3 -m venv .venv
2. Ative o ambiente virtual:
   Windows: .venv\Scripts\activate
   Unix/macOS: source .venv/bin/activate
3. Para desativar o ambiente:
   Windows/Unix/macOS: deactivate

Com o ambiente criado e ativado, basta executar os mesmos comandos para instalar as bibliotecas citados anteriormente.

3.2 Guia de utilização

Com os requisitos mínimos já efetuados, pode-se executar o programa. Ainda, é importante conferir se todos os arquivos de programa estão no diretório junto do arquivo .env. Para executar, basta clicar em “main.py” ou abrir um terminal e chamar este mesmo arquivo (se as bibliotecas foram instaladas localmente é necessário ativar o ambiente virtual).

Assim que a aplicação é iniciada, o usuário é perguntado se deseja construir a base de dados (responder “y” para sim ou “n” para não). Caso esta não exista, esse passo é necessário. Se a base já existir, ao responder afirmativamente novos dados serão acrescentados, sem deletar os antigos. Dependendo do número de músicas, esse processo pode demorar uma vez que é necessário buscar os dados da API via internet. Esse tempo não é longo, em testes feitos com 10 mil músicas (quantidade muito acima da média), o tempo necessário foi de 1 minuto.

Após, as opções de busca serão exibidas:

- Ordenar por popularidade: é possível exibir tanto as 100 músicas mais populares quanto as 100 menos. Para isso, é necessário escolher entre ordem descendente (“y”) ou ascendente (“n”), respectivamente.
- Procurar por nome: é necessário digitar exatamente o nome da música. Isso retorna o registro música completo, com seu ID, nome de artista, duração e explicitude.
- Procurar pro prefixo: basta digitar um prefixo de qualquer música (não é necessário se preocupar com letras maiúsculas ou minúsculas) e o programa buscará todas as músicas com aquele prefixo. Ademais, é necessário escolher a forma de ordenação dos resultados (popularidade, nome da música ou nome do artista).
- Gerar análises: será exibido o nº de músicas, média de popularidade, média de duração e porcentagem de músicas explicítas.

Durante a exibição dos resultados, é possível reordenar a tabela teclando “r”. Isso inverte totalmente a lista de músicas e retorna para a primeira página. Para navegar pela tabela, basta usar as setas para direita (avançar) ou esquerda (voltar).

Por fim, para sair desta busca, é necessário teclar “q”. Após sair de uma pesquisa, o programa pergunta se o usuário deseja fazer mais buscas ou encerrar o programa.

4 CONCLUSÃO

A realização deste trabalho foi considerada de grande aprendizado pela dupla. Apesar das dificuldades e problemas que foram surgindo ao longo do desenvolvimento, esse projeto permitiu a aplicação prática dos conteúdos estudados na disciplina e desenvolvimento de novas habilidades.

4.1 Aprendizados

Principalmente, com este trabalho, aprendeu-se muito a respeito sobre estruturação de arquivos e planejamento de *software*, habilidade de extrema importância no desenvolvimento de aplicações. Com ele, foi possível se aprofundar nas formas de organizar arquivos e suas estruturas auxiliares, como as de índice. O conhecimento adquirido de árvores-B e TRIE's ampliou os horizontes a respeito de estruturas de dados, antes limitados a árvores binárias simples.

Ademais, este projeto foi um grande "treino" de programação em Python, linguagem que agora a dupla passa a dominar mais. De todos os conhecimentos que foram aprimorados, ressaltam-se dois: a programação orientada a objetos e o uso de API's. Este último sendo muito importante uma vez que foi o primeiro contato dos estudantes.

4.2 Dificuldades

Conforme o desenvolvimento do programa, diferentes obstáculos foram surgindo. No início, aprender a usar a API para extração de dados se mostrou um desafio. Isso demandou que a dupla estudasse a documentação detalhadamente para efetivamente poder coletar informações e montar uma base de dados.

Implementar as estruturas em si não foi um grande desafio. No entanto, planejar de antemão como cada entidade iria se relacionar para enfim começar o código não se mostrou uma tarefa fácil. A habilidade de pensar a "engenharia" de uma aplicação foi uma novidade que no início demandou certo esforço, mas que com o tempo foi sendo superada.

4.3 O que poderia ser feito

Conforme esperado, muitas ideias que surgiram antes de começar o desenvolvimento foram alteradas ou descartadas. Apesar de a dupla considerar que satisfatoriamente concluiu as funcionalidades principais do projeto, algumas ideias que agregariam ao resultado final foram descartadas.

A API fornece muitas informações sobre cada música como qual sua velocidade, tom e mudanças de compasso, que dizem muito sobre o gosto musical do usuário. No entanto, apenas dados mais gerais foram de fato coletados. Seria interessante também, permitir que a aplicação pudesse calcular estatísticas a respeito desses dados.

Outra funcionalidade que acabou por não ser adicionada, foi a de usar a B-tree para pesquisar por ID. Este é um atributo único para cada música e sua busca pode ser útil quando o usuário só tem o ID da música e quer saber qual música exatamente aquele ID representa. Além do mais, com a adição da TRIE, usar a B-tree para buscar por nome se tornou uma funcionalidade menos útil.

Por fim, uma ideia mais ambiciosa que foi rapidamente descartada é a de criar uma aplicação *web* a partir deste projeto. Isso possibilitaria usuários de utilizar a aplicação pela internet, sem precisar instalar todos os requisitos necessários.

Apesar dos desafios e das ideias que tiveram que ser deixadas de lado, muitas dessas mudanças podem ser facilmente feitas. O código está bem documentado e permite que no futuro mais funcionalidades possam ser implementadas.

REFERÊNCIAS

Paul Lamere. Spotipy 2.24.0 documentation. Disponível em: <https://spotipy.readthedocs.io/en/2.24.0/>. Acesso em: 10 jul. 2024.

Will McGugan. Rich 13.6.0 documentation. Disponível em: <https://rich.readthedocs.io/en/stable/index.html>. Acesso em: 19 jul. 2024.

Venv - Creation of virtual environments. Python documentation. Disponível em: <https://docs.python.org/3/library/venv.html>. Acesso em: 1 jul. 2024.