Vinícius Guardieiro Sousa

Estudo de caso do uso de um sistema de gerenciamento de bancos de dados vetoriais para a manipulação, análise e recuperação de áudio

Uberlândia-MG

2025

Vinícius Guardieiro Sousa

Estudo de caso do uso de um sistema de gerenciamento de bancos de dados vetoriais para a manipulação, análise e recuperação de áudio

Trabalho apresentado à Faculdade de Ciência da Computação, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Orientador: Profa. Dra. Maria Camila Nardini Barioni

Uberlândia-MG

2025

Vinícius Guardieiro Sousa
Estudo de caso do uso de um sistema de gerenciamento de bancos de dados vetoriais para a manipulação, análise e recuperação de áudio/ Vinícius Guardieiro Sousa. – Uberlândia-MG, 2025-
52p. : il. (algumas color.) ; 30 cm.

Orientador: Profa. Dra. Maria Camila Nardini Barioni

Conclusão de curso – Universidade Federal de Uberlândia – UFU
Faculdade de Ciência da Computação, 2025.

1. Palavra-chave1. 2. Palavra-chave2. 2. Palavra-chave3. I. Orientador. II. Universidade Federal de Uberlândia. III. Faculdade de Ciência da Computação. IV. Título

Vinícius Guardieiro Sousa

Estudo de caso do uso de um sistema de gerenciamento de bancos de dados vetoriais para a manipulação, análise e recuperação de áudio

Trabalho apresentado à Faculdade de Ciência da Computação, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação

Trabalho aprovado. Uberlândia-MG, 07 de maio de 2025:

Profa. Dra. Maria Camila Nardini Barioni
Orientador

Profa. Dra Fabíola Souza Fernandes Pereira
Convidado 1

Prof. Dr. Marcelo de Almeida Maia
Convidado 2

Uberlândia-MG
2025

Agradecimentos

A conclusão deste trabalho representa não apenas o encerramento de uma etapa acadêmica, mas também a síntese de uma trajetória construída com o apoio, incentivo e dedicação de muitas pessoas que me acompanharam ao longo do curso de Ciência da Computação.

Agradeço, primeiramente, aos meus pais, por todo o amor, paciência e apoio incondicional ao longo de cada passo desta caminhada. À minha família, em especial ao meu irmão, por acreditar no meu potencial mesmo diante das maiores dificuldades.

À minha namorada, por ser presença constante, oferecendo apoio emocional, compreensão e motivação nos momentos de cansaço e incerteza.

Aos amigos que fizeram parte dessa jornada, pelas conversas, trocas de experiências, incentivo mútuo e, acima de tudo, pela amizade verdadeira que tanto me fortaleceu.

Agradeço também aos professores do curso, que foram fundamentais para a minha formação acadêmica, sempre dispostos a compartilhar conhecimento e a contribuir para o meu crescimento pessoal e profissional. Em especial, expresso minha sincera gratidão à minha orientadora, cuja orientação criteriosa, dedicação e confiança foram essenciais para a realização deste trabalho.

Resumo

O presente trabalho tem como objetivo demonstrar a forma como um dado complexo (uma música, por exemplo) pode ser armazenado e manipulado por meio de um Sistema de Banco de Dados Vetoriais (SGBDV), com destaque para a consulta por similaridade de músicas. Para isso, foi realizada revisão bibliográfica acerca do tema e sua aplicação. Além disso, foi desenvolvido um exemplo com uma base de dados para ilustrar como o SGBDV pode ser usado para realizar consultas por similaridade. Os resultados obtidos com a avaliação de precisão e da taxa de *recall* para a representação da música mostraram que existe espaço para melhorias. No entanto, foi possível realizar consultas e formas de representação da música no SGBDV, atingindo assim um dos objetivos deste trabalho, o que abre caminho para a exploração futura de análises de dados baseadas em comparações por similaridade integradas com SGBD.

Palavras-chave: Representação de dados. Sistema de banco de dados vetorial. Busca por similaridade.

Abstract

This work aims to demonstrate how a complex like music can be stored and manipulated through a Vector Database System (VDBMS) for querying music similarity. With this objective in mind, a literature review on the subject and its application was carried out. In addition, an example with a database was developed to illustrate how the VDBMS can be used to perform similarity queries. The results obtained from the precision and recall rate evaluation for music representation indicated that there is room for improvement. Nevertheless, it was possible to perform queries and represent music within the VDBMS, thereby achieving one of the objectives of this work and paving the way for future exploration of similarity-based data analysis integrated with DBMS.

Keywords: Data representation. Vector database system. Similarity searching.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDV	Sistema de Gerenciamento de Banco de Dados Vetoriais
Hz	*Hertz*
dB	*decibel*
MP3	*MPEG-1 Audio Layer 3*
MFCC	*Mel Frequency Cepstral Coefficients*
IBM	*International Business Machines Corporation*
IMS	*Information Management System*
SQL	Linguagem de Consulta Estruturada
LLM	*Large Language Model*
KNN	*k-nearest neighbor*

Sumário

1 Introdução

Com os avanços da computação, tópicos relacionados à inteligência artificial e recuperação por similaridade de imagens e de músicas, por exemplo, ganharam destaque (veja (DOUZE et al., 2025)). Isso impulsionou o surgimento de novas formas de armazenar dados, manipulá-los e representá-los.

As músicas, os filmes e as imagens são dados complexos (BARIONI, 2006). A forma como são representados em um banco de dados influencia na velocidade dos processamentos e na facilidade de resolver problemas, como na busca por similaridade. Como resposta a isso, novos sistemas de gerenciamento de banco de dados surgiram, como os sistemas de banco de dados vetoriais (veja, por exemplo, (WANG et al., 2021)).

Compreender a forma como um dado é representado é fundamental para permitir sua recuperação e facilitar suas manipulações, uma vez que esse entendimento pode evitar etapas desnecessárias de manipulação antes da comparação dos dados, como relatado em (TAMBOLI; KOKATE, 2022a). Combinada à representação, definir o modelo de banco de dados para o armazenamento é fundamental, pois cada sistema de gerenciamento de banco de dados apresentará melhor performance para cada problema.

Esse entendimento da representação de dados complexos e armazenamento em um banco de dados é o que possibilita a criação de aplicativos de comparação por similaridade e reprodução de músicas, como o Shazam (WANG, 2003), Dejavu (KAMUNI et al., 2024) e Spotify (TRAN; SWEENEY; LEE, 2019).

Dada a necessidade de representação de dados complexos, esse trabalho de conclusão de curso tem como **objetivo geral** demonstrar a forma como um dado complexo (no contexto desse trabalho, uma música) pode ser representado em um computador utilizando um vetor de atributos extraídos a partir do arquivo (no contexto desse trabalho, um arquivo de áudio).

Além disso, como **objetivos específicos** deste projeto, pode-se destacar o estudo do armazenamento desse dado complexo em um banco de dados vetorial, e das principais formas de recuperá-lo por similaridade. Para esse objetivo, conceitos como Sistema de Gerenciamento de Banco de Dados Vetorial (SGBDV) e extração de atributos de músicas serão fundamentais. Deseja-se também proporcionar com este trabalho de conclusão de curso uma referência para o estudo do tema “SGBDV aplicado a dados complexos”.

A pesquisa aqui descrita serve como uma introdução a conceitos fundamentais ao tratamento de dados complexos e suas manipulações através de representações vetoriais. O estudo que conduziu à escrita desse trabalho de conclusão de curso também forneceu

uma base sólida para que o autor possa continuar seus estudos (principalmente os que envolvem os sistemas de gerenciamento de banco de dados vetoriais) como parte de um projeto de mestrado.

Esse trabalho de conclusão de curso está organizado da seguinte forma: no Capítulo 2, são apresentados conceitos importantes para compreender como uma música é representada em um computador e o funcionamento de um sistema de banco de dados vetorial. O Capítulo 3 traz uma breve exposição de trabalhos correlatos a este, isto é, referências literárias para os problemas de representação de dados por atributos e armazenamento em banco de dados.

No Capítulo 4, encontra-se a descrição dos processos de população de um banco de dados com atributos musicais e dos métodos de recuperação por meio da similaridade utilizando o *PostgreSQL* com a extensão do *pgvector*. O Capítulo 5 contém uma análise dos resultados obtidos das comparações por similaridade com a aplicação de um modelo de avaliação. Por fim, no Capítulo 6 é apresentada uma discussão sobre os resultados deste trabalho, bem como possíveis trabalhos futuros oriundos desse estudo.

2 Fundamentação teórica

Esse capítulo apresenta os principais conceitos teóricos que são utilizados na comparação de músicas por similaridade. Inicialmente, serão apresentados conceitos musicais básicos que permitem a percepção da similaridade das músicas, como melodia, harmonia, ritmo e timbre. Em seguida, será relatado como um arquivo de áudio é representado em um computador para compreensão dos metadados, dos valores contidos no arquivo e dos atributos que podem ser extraídos dele.

Após o conhecimento da representação dos arquivos de áudio, será analisada a forma como podemos representá-los em um banco de dados para permitir sua recuperação e consultas por similaridade. Por fim, para realizar a comparação por similaridade, será necessário compreender as medidas de similaridade.

2.1 Teoria musical

Nesta seção, serão apresentados alguns conceitos básicos da teoria musical, presentes em (MED, 1996). A música é uma combinação de sons simultâneos e sucessivos. Esses sons são ondas sonoras que se propagam no ar e são perceptíveis pelo homem, como ilustrado na Figura 1.

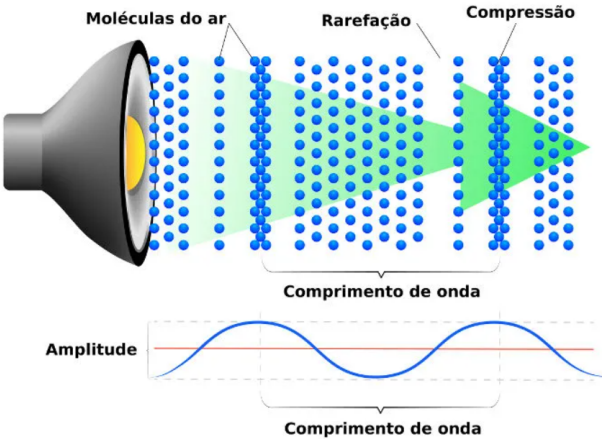

Figura 1 – Propagação de uma onda. Extraído de (HELERBROCK, 2019).

Essas ondas podem ser divididas de duas maneiras: as de vibração regular, que são as notas musicais, e as de vibrações irregulares, que são os ruídos. Para compreender a música, é necessário, primeiro, compreender as características do som, que são: altura, duração, intensidade e timbre.

A altura do som, indicada pela frequência por segundo, *Hertz* (Hz), demonstra a velocidade com que a onda se propaga pelo ar. Quanto maior for a velocidade da onda,

mais agudo será o som. Já a duração indica o tempo em que a onda se propagou. Ela é representada pela unidade de medida de tempo, como milissegundos ou segundos. Outra característica do som é sua intensidade, que indica seu volume (se será forte ou fraco), e é medida em *decibel* (dB). Por fim, o timbre permite distinguir os sons. Por meio dele, pode-se compreender qual instrumento ou pessoa está emitindo o som. Essa característica não apresenta unidade de medida. Alguns exemplos da representação do timbre estão dados na Figura 2.

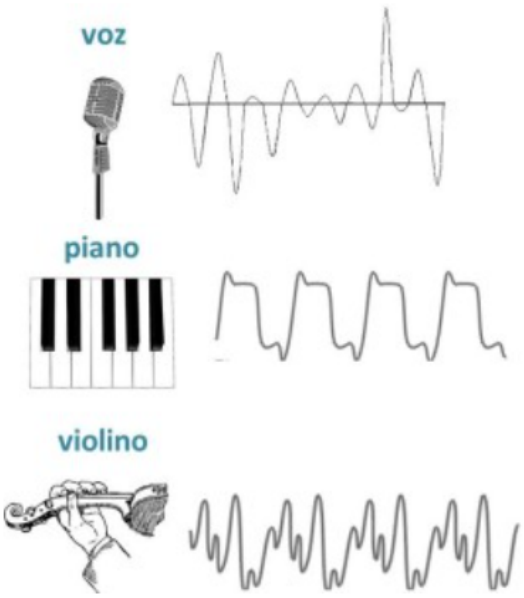

Figura 2 – Representação do timbre. Extraído de (MúSICA, 2023).

Assim como o som, a música apresenta 4 propriedades: a melodia, que é formada pela sucessão de sons; a harmonia, que são sons emitidos ao mesmo tempo, ou seja, a ordem simultânea do som; o contraponto, que são as melodias dispostas de forma harmônica; o ritmo, que é a organização do tempo dos sons.

Esses atributos auxiliam, por exemplo, na identificação de gêneros musicais através da análise de similaridade: analisa-se o timbre, que identifica os instrumentos musicais utilizados, e o ritmo, que dita a forma como a música é tocada. No caso do *Rock* guitarras são muito utilizadas; já no *Jazz*, usa-se o saxofone. Outro exemplo seria comparar o ritmo do Metal, mais agressivo, com o *Reggae*, mais calmo. A análise de gêneros musicais vai além desses atributos, como as letras musicais e a região onde foram criadas.

No trabalho aqui descrito, foram estudados os principais conceitos envolvidos na busca por similaridade de músicas. Também foi desenvolvido um algoritmo de busca que retorna uma lista de músicas similares a uma música dada, o qual foi validado através da análise do gênero musical das respostas fornecidas.

2.2 Representação Computacional de Músicas

Conforme (TANENBAUM; BOS, 2024), os arquivos apresentam dados que auxiliam na compreensão de como foram criados, e informações relevantes sobre eles. Essas características estão situadas no seu metadado. A forma como elas são escritas varia conforme o tipo de arquivo e sua extensão. Arquivos do tipo *.txt* terão uma representação de metadados diferente de arquivos do tipo *.pdf*, por exemplo. O mesmo é válido para arquivos de áudio, que, no nosso contexto, serão do tipo *.mp3*.

Entender a representação de um arquivo de áudio em um computador é fundamental para compreender formas de representá-lo em um banco de dados, o que facilitará sua manipulação, como em uma busca por similaridade. Os conceitos musicais serão necessários para essa análise.

2.2.1 Arquivo de música

Dentro dos arquivos de áudios existentes, será analisado o *MPEG-1 Audio Layer III* (MP3). Esse formato utiliza como padrão de metadado o ID3 (BISPO, 2016). Esse padrão possui duas versões principais: o ID3v1 e o ID3v2. Ambas as versões contêm informações como o título, o artista, o álbum, o ano, alguns comentários e o gênero do áudio. O que as diferencia é a presença dos compositores, da letra da música e da capa do álbum na segunda versão.

A representação de um arquivo de áudio que utiliza o modelo de metadado ID3v1 é realizada em 128 bytes, que estão localizados no final do arquivo e estão dispostos como na Tabela 1.

Campo	Tamanho	Bytes	Descrição
Tag	3 bytes	0-2	Identificador para indicar que o metadado é um ID3v1
Título	30 bytes	3-32	Texto identificando o título do áudio
Artista	30 bytes	33-62	Texto identificando o artista do áudio
Álbum	30 bytes	63-92	Texto identificando o álbum do áudio
Ano	4 bytes	93-96	Texto identificando o ano do áudio
Comentário	30 bytes	97-126	Texto com comentários do áudio
Gênero	1 bytes	127	Número inteiro entre 0 e 255 que mapeia o gênero musical

Tabela 1 – Representação ID3v1. Adaptada de (BISPO, 2016).

Por meio do metadado, é possível coletar alguns atributos do arquivo. Entretanto, existem atributos que são extraídos do próprio arquivo de áudio.

Os demais *bytes* do arquivo são divididos em *frames*, que apresentam um cabeçalho de 4 *bytes* indicando o início do *frame* e informações importantes para a leitura dos demais *bytes*, os dados comprimidos (que são os dados do som, no caso de um arquivo do tipo *.mp3*). Esses dados, quando decodificados, representam o áudio no domínio do tempo discreto da seguinte forma: a cada unidade de tempo associa-se a amplitude do som.

2.2.2 Atributos

Como mencionado anteriormente, pode-se extrair atributos dos arquivos de áudio. Alguns deles podem ser obtidos diretamente do metadado, enquanto outros são extraídos através dos dados da música, conforme demonstrado em (KNEES; SCHEDL, 2013). Estes últimos atributos podem ser subdivididos em dois grupos: os de alto nível e os de baixo nível.

Alguns atributos da música são extraídos no domínio do tempo, enquanto outros estão relacionados ao domínio da frequência. Para realizar a conversão dos domínios é importante conhecer a *Transformada de Fourier*.

A *transformada de Fourier* de uma função $f(t)$ no domínio do tempo discreto é a função $F(w)$ no domínio da frequência dada por um somatório infinito de senoides complexas conforme a Equação (2.1) (DINIZ; SILVA; NETTO, 2014).

$$F(w) = \mathscr{F}\{f(t)\} = \int_{-\infty}^{\infty} f(t)e^{-iwt}dt. \tag{2.1}$$

2.2.3 Atributos de baixo nível

Conforme (KNEES; SCHEDL, 2013), os atributos de baixo nível não necessitam de muitas manipulações para serem extraídos, pois são obtidos com manipulações simples do arquivo de áudio no domínio do tempo ou da frequência. No último caso, a manipulação ocorre após a utilização da Transformada de Fourier em dados correspondentes no domínio do tempo.

A Figura 3 foi gerada utilizando-se a linguagem de computação *Python*. Ela ilustra como é a representação de um arquivo de uma música no computador: uma função que relaciona a amplitude da onda a cada unidade de tempo.

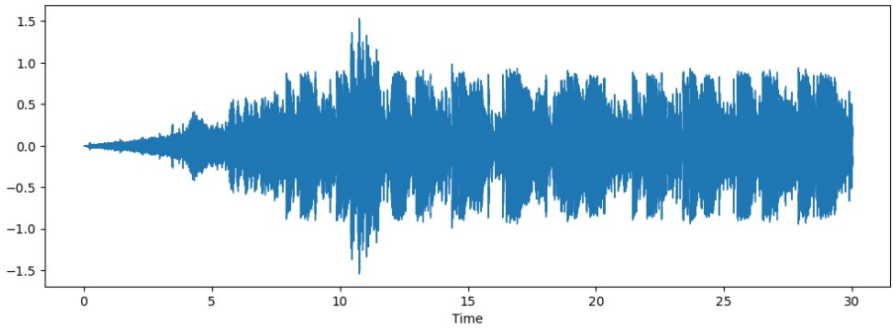

Figura 3 – Gráfico de amplitude x tempo. Fonte do autor.

Por meio desse gráfico, são realizados os cálculos para a obtenção dos seguintes atributos de baixo nível no domínio do tempo:

- *Amplitude Envelope* - representa a amplitude máxima de cada *frame* do áudio. Sua extração é feita calculando-se a maior amplitude de cada *frame*, e é obtida através da Equação (2.2).

$$AE_t = \max_{k=t\cdot K}^{(t+1)\cdot K-1} s(k), \tag{2.2}$$

  onde K denota o tamanho do *frame*.

- *Root-Mean-Square Energy* - caracteriza a intensidade sonora. É menos sensível a ruídos, pois aplica-se a raiz quadrada na média ponderada das amplitudes conforme a Equação (2.3).

$$RMS_t = \sqrt{\frac{1}{K} \cdot \sum_{k=t\cdot K}^{(t+1)\cdot K-1} s(k)^2}. \tag{2.3}$$

- *Zero-Crossing Rate* - utilizada para reconhecimento de fala e ruídos. Para obter esse atributo, basta quantificar quantas vezes a amplitude mudou de sinal (de positivo para negativo ou vice-versa) dentro de uma janela. É dada pela Equação (2.4).

$$ZCR_t = \frac{1}{2} \cdot \sum_{k=t\cdot K}^{(t+1)\cdot K-1} |\text{sgn}(s(k)) - \text{sgn}(s(k+1))|. \tag{2.4}$$

Após a aplicação da *Transformada de Fourier* na função exibida na Figura 3, obtém-se a Figura 4.

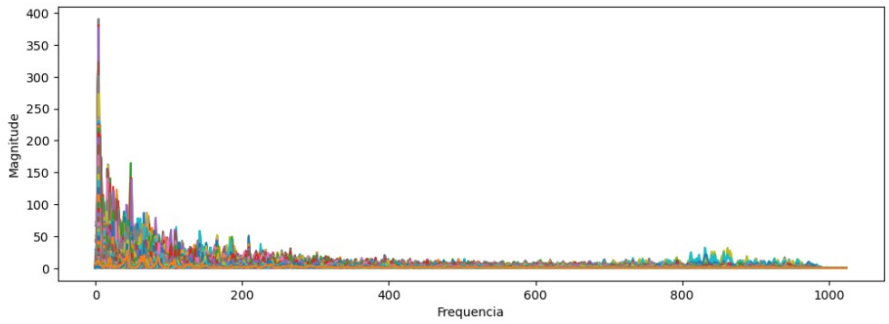

Figura 4 – Gráfico de magnitude x frequência. Fonte do autor.

Novamente, pode-se utilizar esse gráfico para se obter os seguintes atributos de baixo nível no domínio da frequência:

- *Band Energy Ratio* - esse atributo serve para diferenciar fala e músicas, e, associado a outros atributos, auxilia na identificação de gêneros musicais. Para obter esse atributo, basta definir um valor de frequência limite F e dividir o quadrado das magnitude de frequência inferior ao limite fixado com o quadrado das magnitudes das frequências superiores. É dado pela Equação (2.5).

$$BER_t = \frac{\sum_{n=1}^{F-1} m_t(n)^2}{\sum_{n=F}^{N} m_t(n)^2}. \tag{2.5}$$

- *Spectral Centroid* - relacionado ao timbre do som, indica o centro de gravidade da magnitude. Para calculá-lo, basta realizar o somatório de todas as magnitudes multiplicadas por sua frequência e, por fim, dividir esse somatório pelo somatório das magnitudes, conforme a Equação (2.6).

$$SC_t = \frac{\sum_{n=1}^{N} m_t(n) \cdot n}{\sum_{n=1}^{N} m_t(n)}. \tag{2.6}$$

- *Bandwidth* ou *Spectral Spread* - depende do *Spectral Centroid* e serve para descrever o timbre. Ela analisa a região próxima do centro da gravidade da magnitude. Seu cálculo é semelhante ao do *Spectral Centroid*, a diferença é que o somatório do numerador será o módulo da subtração entre a frequência e o *Spectral Centroid* multiplicado pela magnitude. É dado pela Equação (2.7).

$$BW_t = \frac{\sum_{n=1}^{N} |n - SC_t| \cdot m_t(n)}{\sum_{n=1}^{N} m_t(n)}. \tag{2.7}$$

- *Spectral Flux* - utilizado para identificar falas e timbre de músicas, ele analisa a mudança de potência entre quadros consecutivos. Para calcular esse atributo, é necessário ter a distribuição de frequência normalizada de cada quadro e realizar o somatório da diferença quadrática da distribuição de um quadro com seu anterior, conforme Equação (2.8).

$$SF_t = \sum_{n=1}^{N} (D_t(n) - D_{t-1}(n))^2. \tag{2.8}$$

2.2.4 Atributos de alto nível - *Mel Frequency Cepstral Coefficients*

Os atributos de alto nível são extraídos com maior complexidade. Essa extração leva em consideração a percepção humana, tentando aproximar ao máximo os cálculos com o que é perceptível. De acordo com (TZANETAKIS, 2002), alguns exemplos desses atributos são a medição física da intensidade sonora, a medição perceptual do volume, a percepção da frequência e os *Mel Frequency Cepstral Coefficients*. Esse último foi o foco deste estudo.

Os *Mel Frequency Cepstral Coefficients* são utilizados para reconhecimento de falas e como uma forma de modelar o timbre de músicas. É representado por um vetor que relata a periodicidade encontrada no gráfico da magnitude do áudio pela frequência. Ele realiza operações no domínio do tempo e da frequência, porém o retorno do algoritmo ocorre no domínio do tempo.

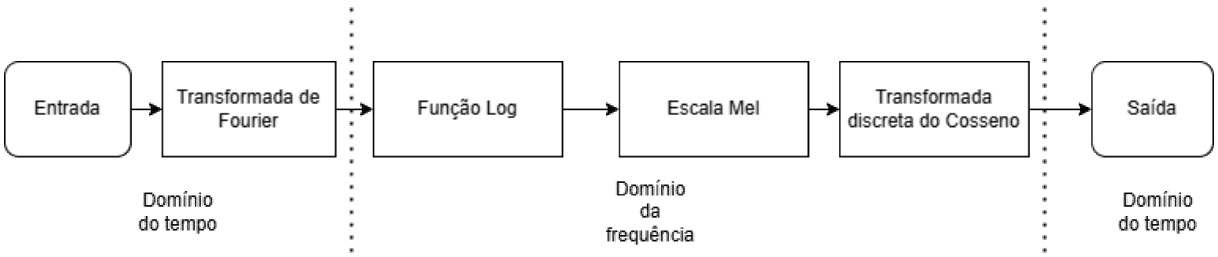

Figura 5 – Algoritmo do *Mel Frequency Cepstral Coefficients*. Adaptado de (DERUTY, 2022).

Conforme demonstrado na Figura 5, no algoritmo do MFCC a entrada será um áudio no domínio do tempo. Na primeira etapa, a transformada de Fourier é aplicada à entrada para convertê-lo para o domínio da frequência, e em seguida aplica-se a função Log no resultado. Após isso, a *escala de Mel* é aplicada e, em sequência, a transformada discreta do cosseno retorna vetores no domínio do tempo como o resultado do algoritmo.

2.2.5 Criação do vetor de atributos

Após compreender a forma de extrair os atributos, é necessário armazená-los como um vetor, o que permitirá aplicar fórmulas para a comparação por similaridade.

No caso de arquivos de áudio, de acordo com (BARIONI, 2006), existem duas técnicas para a criação do vetor. A primeira delas é útil para análise de dados que sofrem atualizações em tempo real, como em rádios. Nessa técnica, o áudio é segregado em pequenos segmentos de tempo e os atributos são calculados para cada segmento. Ao final, haverá um vetor de atributos para cada segmento. A segunda técnica é mais utilizada para dados que não sofrem atualizações, como em um arquivo de música. Nesse cenário, um único vetor é calculado para toda a música.

2.3 Medidas de Similaridade

As medidas de similaridade dependem da maneira como o problema é modelado. Alguns exemplos de verificação de similaridade são: por conteúdo do áudio, por metadados e por vetor de atributos. Essas formas de verificação podem ser combinadas para melhorar o resultado obtido. O foco desse trabalho foi a busca por similaridade por vetor de atributos.

2.3.1 Métricas de comparação

Para metrificar o quão semelhantes são duas músicas, é necessário validar o quão próximos seus vetores de atributos são. Para isso, métricas de distância entre vetores, como *Norma Lp* e *Distância Cosseno* podem ser utilizadas (veja (KNEES; SCHEDL, 2013)). Para se definir a Norma Lp, necessita-se de um número $p \geq 1$, que representa o tamanho do vetor atributo. A distância entre dois vetores x e y induzida pela norma Lp é então dada pela Equação (2.9).

$$||x - y||_p := \left(\sum_i^n |x_i - y_i|^p \right)^{1/p}. \tag{2.9}$$

Outra forma de se metrificar quão distantes os vetores de atributos x e y são é utilizar a Distância Cosseno, que é dada pela Equação (2.10).

$$d(x, y) \mapsto 1 - \frac{x \cdot y}{||x||_2 ||y||_2} = 1 - \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}}. \tag{2.10}$$

Quanto menor a distância entre os vetores de atributos x e y, maior a similaridade entre as músicas.

2.3.2 Avaliação da busca

Após a criação de um algoritmo de busca por similaridades, é importante verificar sua eficácia. Uma forma de avaliar se esse algoritmo apresenta resultados satisfatórios é utilizar funções como a **precisão**, a **taxa de *recall*** e a ***F-measure***, semelhante ao que foi realizado em (KNEES; SCHEDL, 2013). Para utilizar essas métricas, é preciso definir quais dados são relevantes na comparação de similaridade. Por exemplo, pode-se analisar a eficácia de um algoritmo avaliando-se o gênero das músicas retornadas por ele, em comparação com a música utilizada como entrada.

Nesse exemplo, a taxa de precisão do algoritmo é obtida ao se dividir o número de verdadeiros positivos (músicas do mesmo gênero que foram retornadas pelo código) pelo número de resultados retornados. Quanto menor a precisão, maior a quantidade de falsos positivos, ou seja, músicas de gêneros distintos do desejado retornadas.

Isso significa que o algoritmo entende como similares músicas que, na realidade, não o são. Se um algoritmo possuir baixa taxa de precisão, deve-se melhorar os atributos que representam a música. A taxa de precisão P é dada pela Equação (2.11).

$$P = \frac{|Rel \bigcap Ret|}{|Ret|}, \tag{2.11}$$

onde Rel denota o conjunto de resultados relevantes no banco de dados e Ret denota o conjunto de resultados retornados pelo algoritmo.

Para se obter a taxa de *recall*, divide-se o número de verdadeiros positivos pela quantidade total de músicas do mesmo gênero no banco de dados. Quanto menor esse valor, maior a quantidade de músicas "desejadas" que não foram retornadas pelo algoritmo. A taxa de recall R é dada pela Equação (2.12).

$$R = \frac{|Rel \bigcap Ret|}{|Rel|}. \tag{2.12}$$

Se um algoritmo possuir baixa taxa de *recall*, deve-se melhorar a forma de representar a música.

A fórmula da *F-measure* utiliza a precisão da Equação (2.11) e a taxa de *recall* da Equação (2.12). Quanto menor uma dessas taxas, menor será o resultado dessa métrica. A taxa de F-measure F é dada pela equação (2.13).

$$F = 2 \cdot \frac{P \cdot R}{P + R}. \tag{2.13}$$

Utilizar apenas uma métrica (precisão ou taxa de *recall*) pode gerar uma falsa impressão de que a busca por similaridade atendeu o esperado. Por isso, é interessante analisar essas taxas simultaneamente, para compreender se o algoritmo retorna muitos falsos positivos ou se ele deixa de exibir muitos resultados esperados.

2.4 Armazenamento em Banco de Dados

O sistema de gerenciamento de banco de dados tem como objetivo facilitar a manipulação dos dados, auxiliando nas necessidades do problema a ser resolvido. Por isso, cada arquitetura de SGBD terá suas particularidades, que serão vantajosas em relação às outras.

2.4.1 Sistema de Gerenciamento de Bancos de Dados Relacional

O SGBD relacional foi desenvolvido em 1970 pelo pesquisador Edgar Frank Codd da IBM, (RAMAKRISHNAN; GEHRKE, 2003). Nesse modelo, os dados são organizados de maneira pré-definida, eles são dispostos em forma de tabelas, conforme demonstrado em (ANDRADE, 2024).

Esse formato de tabelas pode ser representado por meio de um diagrama de relação de entidades. Essa representação é útil para facilitar o entendimento de como cada tabela relacionam-se e o que pode conter em cada uma delas, facilitando a criação de *querys* para a manipulação dos dados.

Esse modelo tem como vantagem sua relação entre as tabelas, garantindo um relacionamento lógico entre os dados. Além disso, nesse modelo, há uma garantia de integridade referencial nos dados por meio do uso de chaves primárias e estrangeiras (RAMAKRISHNAN; GEHRKE, 2003). Outra vantagem está na facilidade de realizar consultas e na sua padronização.

Entretanto, esse modelo de banco de dados não é muito flexível, conforme relatado em (RODRIGUES, 2017), ou seja, é complexo alterar os modelos relacionais quando um banco de dados está em uso. Além disso, ele não é performático para dados não estruturados como imagens e música.

2.4.2 Sistema de gerenciamento de bancos de dados não relacionais

Com os avanços tecnológicos, a quantidade de dados processados e o poder computacional aumentaram. Com isso, o conceito da computação distribuída surgiu e a estrutura dos modelos relacionais não estava atendendo suas necessidades devido ao modelo relacional ser mais rígido (veja (RODRIGUES, 2017)). Seu principal objetivo era trazer maior flexibilidade na manipulação de dados, os quais podem seguir o modelo orientado a documentos, cujas vantagens são a flexibilidade de esquema e a armazenagem de dados complexos como vetores.

2.4.3 Sistema de gerenciamento de bancos de dados vetoriais

Nessa seção serão apresentados conceitos fundamentais para o Sistema de Gerenciamento de Banco de Dados Vetoriais presentes em (PAN; WANG; LI, 2023).

Os dados salvos em vetores são muito importantes para sistemas de recomendação e busca por músicas semelhantes. Por causa disso, pesquisas referentes a um sistema de gerenciamento de banco de dados dedicado a vetores surgiram, mesmo com a existência do modelo não relacional que facilita o armazenamento de vetores, pois manipulações mais complexas sobre vetores serão necessárias (como no estudo da similaridade de vetores), (TAIPALUS, 2024).

A vantagem de utilizar um SGBDV para representar vetores está na preocupação com a performance desse sistema para manipular vetores. Essa manipulação engloba algoritmos para realizar consultas. Já a performance das manipulações dependerá do modelo de indexação utilizado.

Os Sistemas de Gerenciamento de Banco de Dados vetorial pode ser puramente vetorial ou com múltiplas representações (onde modelos relacionais e não relacionais também são empregados). Conforme (LTD, 2013), um modelo puramente vetorial é o *Pinecone* e *Milvus*, já o *MongoDB* é um exemplo de SGBD não relacional que dá suporte ao modelo vetorial. Um SDGBs relacionais com suporte ao modelo vetorial é o *PostgreSQL* com a extensão *pgvector*.

2.4.3.1 Tipos de Consultas

Para consulta de dados tradicionais (como números e textos), normalmente utiliza-se operadores matemáticos, como igualdade, diferenciação ou menor que. Porém, em dados complexos representados por vetores, o foco é na similaridade entre eles. Para isso, utilizam-se cálculos de distância.

Existem diferentes estratégias de consultas por similaridade, por exemplo: consultas de busca simples (*basic search queries*) e consultas multivetoriais (*multi-vector queries*). Na estratégia de busca simples, realiza-se o cálculo da distância entre dois vetores, conforme apresentado na Seção 2.3.1.

Já a consulta multivetorial será útil para quando cada item de dado é representado por um conjunto de vetores, como em um reconhecimento facial. Como não é o caso desse trabalho de conclusão de curso, a estratégia da busca simples utilizando o cálculo de distância Lp e Cosseno será utilizada.

Considerando as consultas de busca simples, existem diferentes tipos de consultas, por exemplo: consulta aos k-vizinhos mais próximos (k-*nearest neighbor* - KNN) e consulta por abrangência (*range query*).

Definição 1 *Encontra um subconjunto* $S1$ *de* S*,* $S1 \subset S$*, de tamanho* k*, sendo*$d(X', q) \leq c(\ min_{x \in S}\, d(\boldsymbol{x}, \boldsymbol{q}))$ *para todo* $X' \in S'$*.*

O objetivo do KNN, definido em Definição 1, é obter, dentro de um conjunto K elementos com a menor distância de um dado elemento. Com esse algoritmo, se k for menor que a quantidade de elementos para a comparação, o resultado sempre será um subconjunto com k elementos, não importando se a distância entre os vetores são grandes, basta que ela seja a menor dentro do grupo, sendo uma escolha para criação de *ranking* por similaridade ou quando deseja uma quantidade fixa de elementos.

Definição 2 *Encontra um subconjunto* $S1$ *de* S*,* $S1 \subset S$*, onde, para todo* $X \in S1, d(X, q) \leq r$*, sendo* q *o dado comparado e* r *um limiar para o tamanho do raio.*

Por outro lado, a busca por abrangência, ou *range query*, definida em Definição 2, leva em consideração o tamanho da distância entre os dados comparados. Com ela, pode-se retornar um conjunto vazio ou o próprio conjunto que foi comparado. Como ocorre o retorno de todos os elementos cuja distância seja menor que a de um limiar definido.

2.4.3.2 Indexação

Utilizar o algoritmo KNN para comparar um vetor com todos os vetores do banco de dados demanda um alto poder computacional, pois conforme os dados aumentam a quantidade de comparação também aumenta, sendo $O(ND)$, onde N é o custo do KNN e D a quantidade de vetores no banco de dados. Uma forma de melhorar a performance para a consulta é reduzindo a quantidade de vetores utilizados no momento da comparação.

Para gerar um subconjunto de vetores para realizar a comparação é necessário aplicar uma estratégia de indexação, e para cada tipo de problema existirá uma que melhor satisfaz, podendo ser:

- *Product Quantization*: ocorre a divisão dos vetores em pequenas partes. É útil para busca de imagens, uma vez que reduz a dimensionalidade do vetor, porém, em compressões com perda, a acurácia será prejudicada.
- *Locality-Sensitive Hashing*: cria *hashes* de vetores similares para o mesmo *bucket*. É útil para detecção de duplicidade próxima, pois permite a busca por similaridade aproximada, porém requer alguns ajustes de parâmetros.
- *Hierarchical Navigable Small World*: cria um gráfico hierárquico. Pode ser utilizado em sistemas de recomendação e busca em texto, apresenta alta velocidade para exploração de vizinhos, porém é uma estrutura complexa.

Realizando essa indexação a performance melhora, porém, a consulta será aproximada.

2.5 Considerações Finais

A extração de atributos para representar um arquivo de áudio não é trivial, uma vez que são necessárias manipulações matemáticas para obtê-los de acordo com o objetivo da comparação por similaridade. O conjunto de atributos de uma música é a forma de representá-la computacionalmente, que pode ser feita através de um vetor de atributos. Com isso, para realizar a comparação por similaridade, algoritmos de medida de distância de vetores podem ser utilizados.

Após a definição dos atributos, que serão extraídos por meio do MFCC, é importante definir qual o SGBD que será utilizado, no caso desse trabalho de conclusão de curso, o SGBD será o *PostgreSQL* com a extensão do *pgvector*. Isso se deve pelo fato de aproveitar as vantagens de dois modelos, o relacional e o vetorial. Do SGBD relacional tem-se as vantagens da integridade referencial e da facilidade para análise de dados, já do SGBDV, as vantagens serão relacionadas a performance e facilidade na manipulação de vetores.

3 Trabalhos Correlatos

É possível encontrar na literatura trabalhos que abordam a comparação por similaridade, os quais também envolvem a extração de atributos e sua representação no banco de dados. Nesse capítulo, alguns desses trabalhos serão descritos.

3.1 Representação por atributos

A busca por similaridade em músicas pode ser utilizada para diversas soluções, como para encontrar uma música por meio de um trecho ou para recomendações musicais. Por exemplo, em (RAZLOGOVA, 2018), *Avery Li-Chun Wang* relata como funciona seu mecanismo capaz de identificar músicas que são reproduzidas em rádio, o *Shazam*. Nesse algoritmo, é criado um identificador para cada música, um *hash*, que serve como uma impressão digital para o arquivo. Esse identificador não depende do trecho que está sendo analisado, o que permite recuperar músicas de forma eficiente.

Esse modelo utilizado pelo *Shazam* requer uma representação complexa da música para permitir a recuperação por trechos não especificados. Por meio dele, é possível realizar buscas por similaridade, pois o identificador é criado com base no espectrograma do arquivo de áudio, porém seu armazenamento não será trivial. Por isso, outras maneiras de representar as músicas são levadas em consideração.

Outra maneira de se realizar a representação é por meio do MFCC, conceito utilizado por diversos trabalhos referentes à comparação por similaridade. Trabalhos como (LAUREANO et al., 2022) e (FACHINI; HEINEN, 2016) utilizam esse algoritmo para resolver os problemas propostos. O primeiro trabalho focou em identificar as notas musicais tocadas por diversos instrumentos, enquanto o objetivo do segundo trabalho foi identificar o instrumento musical, sendo necessário analisar o timbre para a validação.

É comum o uso do MFCC quando se deseja analisar o timbre, e essa representação se assemelha ao sistema auditivo humano, conforme relatado em (BORJIAN, 2017). Porém, para comparar semelhança entre gêneros musicais, outros atributos podem ser utilizados, , como o ritmo. Alguns estudos, como (NAM, 2012), tendem a utilizar aprendizado de máquina para realizar a representação da música por meio de atributos e realizar a classificação por gênero. Além disso, o uso de redes neurais artificiais adaptativas foram estudadas por (TAMBOLI; KOKATE, 2022b) para a busca por similaridade de gêneros musicais, porém a representação da música, nesse trabalho é dinâmica.

Uma outra maneira seria por meio da combinação de algumas transformadas e redes neurais convolucionais, como feito em (SHUVAEV; GIAFFAR; KOULAKOV, 2017).

Nessa técnica, a entrada para a rede neural é a transformada de Fourier de curto prazo, o logaritmo dessa transformada e a transformada de matriz aleatória.

Trabalhos recentes buscam o uso de *Large Language Models*, um modelo de inteligência artificial para aprender a estrutura da linguagem, para resolver problemas como sistema de recomendação de músicas, (YUN; LIM, 2025). Outros usos dos LLMs é para obter significado semântico nas músicas, como em (TAIEF, 2024). Com os avanços dos modelos artificiais torna-se possível representações das músicas de forma aprendida.

Este trabalho teve como foco o uso do MFCC para buscar similaridade entre as músicas e avaliar se o algoritmo desenvolvido foi um bom sistema de indicação de músicas de mesmo gênero.

3.2 Armazenamento no banco de dados

A história do SGBD se iniciou em 1960, com o armazenamento de dados integrados. Porém, com a necessidade de outras formas de armazenar os dados, a IBM, em 1970, criou o modelo relacional de representação de dados, dando origem ao Sistema de Banco de Dados Relacional (veja (RAMAKRISHNAN; GEHRKE, 2003)). Conforme a computação evoluiu, novas necessidades de manipular os dados surgiram, como para a computação distribuída, e, consequentemente, surgiu um novo modelo de banco de dados, o modelo de banco de dados não relacional (veja (ANDRADE, 2024)).

Da mesma maneira, surgiu a preocupação em como armazenar dados complexos e ter um sistema que fosse otimizado para essa solução, pois desafios como a busca de vizinhos mais próximos e avanços na inteligência artificial dependiam de armazenamento e manipulação de vetores (TAIPALUS, 2024). Com isso, alguns pesquisadores começaram a procurar formas de representar os dados para ter buscas eficientes, como o desenvolvimento de uma estrutura de dados baseada em grafos para buscas de vizinhos mais próximos, conforme relatado em (MALKOV; YASHUNIN, 2018).

Essa necessidade de manipulação de vetores com grandes dimensões também foi notada em (DOUZE et al., 2025) (WANG et al., 2021). Isso se deve ao fato da maioria dos dados ser não estruturados, como imagens, vídeos e textos, além de mencionarem o crescimento da Inteligência Artificial. Para conseguir trabalhar com vetores, eram necessárias alterações e a criação de extensões das linguagens, como a *SQL*, para implementar tanto a representação quanto as funções que permitem calcular a similaridade dos dados, como relatado em (BARIONI, 2006).

Com isso, surgiram bibliotecas, como a Faiss (DOUZE et al., 2025), para auxiliar a criação da representação de modelos vetoriais e sistemas de gerenciamento de bancos de dados vetoriais exclusivos, como o Milvus (WANG et al., 2021), que permite a mani-

pulação e armazenamento de vetores. Outros SGBDs que apresentam múltiplos modelos de representação, como o *PostgreSQL*, também aderiram ao modelo vetorial. Nesse caso, para usá-lo, é necessário instalar uma extensão desenvolvida para esse modelo.

Essa extensão é o *PGVECTOR*, que apresenta diversos arquivos da linguagem C com cabeçalhos para permitir a representação do vetor. Além disso, algumas funções de distância já estão implementadas, facilitando a busca por meio dessa distância dos vetores armazenados. A forma como essa extensão foi criada não é trivial, sendo necessário conhecimento em extensões para o *Postgres* e em banco de dados relacional. Porém, sua utilização foi simplificada, permitindo que, com pequenos comandos SQL, tenha-se uma tabela que armazena vetores e realize buscas calculando a distância entre eles.

Apesar dos diversos sistemas de banco de dados vetoriais existentes, o foco desse trabalho de conclusão de curso será o *PostgreSQL* com sua extensão, *pgvector*. Utilizar modelos relacionais com a extensão de modelos vetoriais será vantajoso, pois combina os benefícios, como facilidade para análise e gerenciar relatórios, conforme mencionado em (ANDRADE, 2024), dos SGBDs relacionais com a facilidade de manipular dados vetoriais.

4 Método de trabalho

Nesse trabalho de conclusão de curso, o SGBD vetorial será utilizado para a armazenagem e consulta de músicas por similaridade, conforme a Figura 6. Para a etapa de inserção de dados, primeiro ocorrerá a extração do vetor de atributos da música de entrada, que, por meio da inserção, será indexada no banco de dados seguindo o algoritmo de indexação que ele implementa. Na etapa de consulta por similaridade, a música de entrada também passa pela etapa de extração do vetor de atributos e, por meio de um comando de consulta, esse vetor também passa pela indexação para retornar seus semelhantes, porém, nas consultas, esse vetor não será salvo.

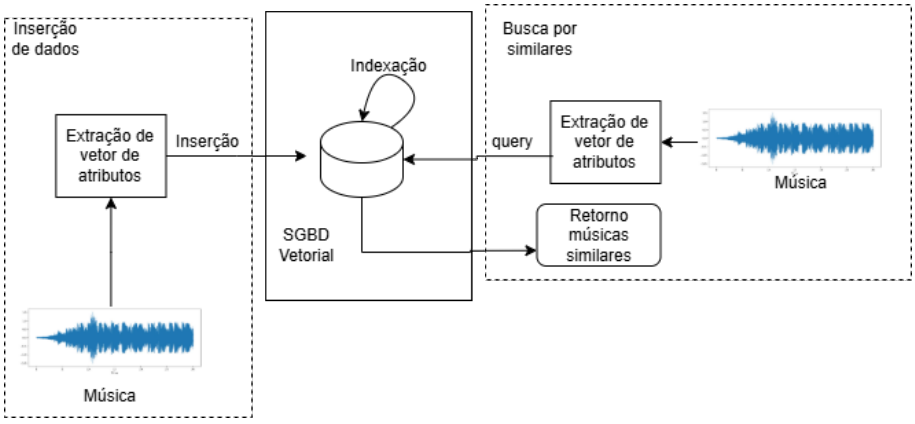

Figura 6 – Representação de um SGBD Vetorial. Imagem adaptada (TAIPALUS, 2024).

Utilizando o *PostgreSQL* com sua extensão para suporte a dados vetoriais, a música poderá ser armazenada como um modelo relacional e seu vetor de atributos se aproveitará dos benefícios de um SGBD vetorial. Nesse caso, dados como o título da música, gênero e artista serão salvos utilizando o modelo relacional, já os vetores de atributos que são utilizados para a busca por similaridade serão indexados e contarão com manipulações dentro do próprio banco de dados, como o cálculo de distância de vetores.

Com essa implementação, será possível a obtenção de um *ranking* por similaridade de músicas por meio do id da música, pelo título ou por um arquivo de áudio. Para atingir esse objetivo, inicialmente fez-se necessário o estudo do tema (descrito no Capítulo 2), a coleta, a modelagem do banco de dados, a inserção e a recuperação de dados, como ilustrado na Figura 7.

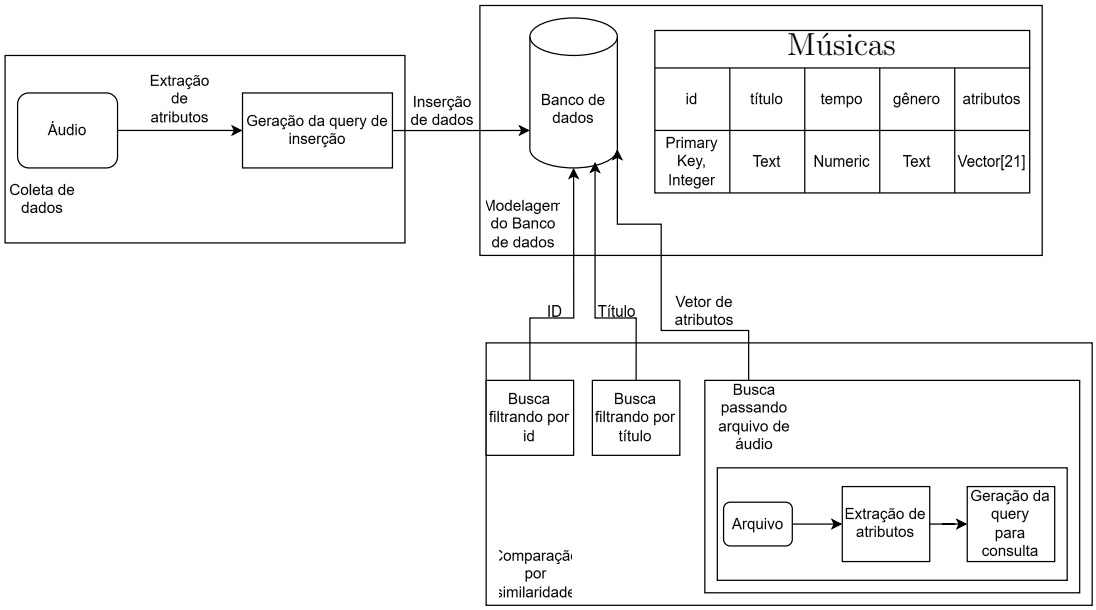

Figura 7 – Etapas realizadas no trabalho. Fonte do autor.

4.1 Coleta de dados

Nessa etapa, foi necessário obter músicas para serem manipuladas nas demais etapas. Elas podem ser obtidas por meio do download em bibliotecas de músicas gratuitas ou pagas, como o *artlist* (BELSKY, 2015), o *Youtube* (HURLEY, 2005) e o *kaggle* (GOLDBLOOM, 2010).

A base de dados *Music Features* do *Kaggle* (HAJOORI, 2018a) foi utilizada para popular um banco de dados para testes com poucos dados, divididos em 10 gêneros e 100 músicas por gênero. O criador dessa base não disponibilizou os arquivos para a extração de atributos, retornando apenas uma planilha com as informações necessárias para a comparação por similaridade, são eles:

- *filename* - representa o nome do arquivo.
- tempo - tempo do áudio.
- *label* - gênero do áudio (*reggae*, clássico, *hiphop*, *disco*, *jazz*, *pop*, *country*, *blues*, *rock* ou *metal*).
- *beats* - pulsos rítmicos da música
- *mfcc1, mfcc2 ,mfcc3, mfcc4, mfcc5, mfcc6, mfcc7, mfcc8, mfcc9, mfcc10, mfcc11, mfcc12, mfcc13, mfcc14, mfcc15, mfcc16, mfcc17, mfcc18, mfcc19, mfcc20* - atributos extraídos por meio do Algoritmo descrito na Seção 2.2.4.
- outros atributos, como: *chroma_stft, rmse, spectral_centroid, spectral_bandwidth, rolloff, zero_crossing_rate*

Nas descrições dessa base de dados, o autor disponibilizou o código utilizado (HAJOORI, 2018b) na extração de atributos para gerar a tabela completa.

4.2 Extração de atributos

Para essa etapa, o autor da base de dados utilizada realizou a leitura de cada áudio e a extração de seus atributos por meio da biblioteca Librosa (MCFEE et al., 2025), conforme o Algoritmo 1.

Algoritmo 1: Extração de atributos por meio da biblioteca librosa

Input: Path do arquivo de áudio

Algoritmo para extrair os atributos de um arquivo de áudio
$y, sr \leftarrow librosa.load(input, mono = True, duration = 30)$
$tempo, beats \leftarrow librosa.beat.beat\_track(y = y, sr = sr)$
$chroma\_stft \leftarrow librosa.feature.chroma\_stft(y = y, sr = sr)$
$rmse \leftarrow librosa.feature.rmse(y = y)$
$spec\_cent \leftarrow librosa.feature.spectral\_centroid(y = y, sr = sr)$
$spec\_bw \leftarrow librosa.feature.spectral\_bandwidth(y = y, sr = sr)$
$rolloff \leftarrow librosa.feature.spectral\_rolloff(y = y, sr = sr)$
$zcr \leftarrow librosa.feature.zero\_crossing\_rate(y)$
$mfcc \leftarrow librosa.feature.mfcc(y = y, sr = sr)$

Por meio da biblioteca *Librosa*, executou-se a leitura de 30 segundos da música com a função *librosa.load*, sendo que y indica os bytes da música e sr o *sample rate* (que indica o número de vezes que a amostra de som original é registrada por segundo). Essas informações são usadas como parâmetros das funções para extração de atributos, sendo que cada atributo tem sua função específica e, consequentemente, um retorno específico.

Com exceção dos atributos $mfcc1, \ldots, mfcc20$, o resultado retornado pelo algoritmo acima foi um vetor com valores para cada *frame* da música. Para se obter apenas um valor por atributo, o autor da base de dados utilizou uma outra biblioteca comum em *Python*, a NumPy (OLIPHANT, 2005). Com ela, cada vetor é transformado em apenas um número por meio da média ($numpy.mean(vector)$), e os valores são concatenados em um arquivo *.csv*.

Para a obtenção dos atributos $mfcc1, \ldots, mfcc20$, utilizou-se o comando

$$mfcc = librosa.feature.mfcc(y = y, sr = sr).$$

Esse comando retorna um vetor de tamanho 20, em que cada entrada é um vetor cujo tamanho varia conforme a quantidade de *frames*. O atributo $mfcc1$ foi então definido como sendo a média dos valores da primeira posição do vetor $mfcc$, e assim sucessivamente.

Com isso, foi possível obter uma base de dados de 1000 arquivos de áudios em formato de tabela com os atributos extraídos, facilitando assim a implementação do banco de dados e o desenvolvimento deste trabalho de conclusão de curso.

4.3 Modelagem do banco de dados

Para essa etapa foi necessário conhecer as informações que poderiam ser extraídas da base de dados. Com isso, foi possível modelar o banco de dados para que, nas consultas, os dados desejados fossem retornados. Com a base disponibilizada em (HAJOORI, 2018a), notou-se que seria possível recuperar o título, o tempo, o gênero e alguns atributos.

Dessa maneira, por se tratar de um banco de dados relacional, como relatado na Seção 2.4, os dados utilizados serão representados em uma forma de tabela (conforme ilustra a Figura 7). No banco de dados, para cada atributo de uma tabela são definidos tipos de dados de acordo com seu domínio. Os campos da tabela com seus respectivos tipos de dado são:

- Id - Uma chave primária (*Primary Key)* do tipo inteiro com incremento automático conforme dados são inseridos. Por se tratar de uma chave primária, é um campo com valor diferente para cada dado.
- Título - Campo do tipo *Text* destinado para o nome do arquivo que foi utilizado para a extração dos atributos.
- Tempo - Campo do tipo numérico para armazenar o tempo do arquivo de áudio.
- Gênero - Campo do tipo *Text* que armazena o gênero da música utilizada.
- Atributos - Campo de vetor, criado com a extensão *pgvector*, para armazenar os atributos extraídos do áudio, como o *MFCC* e *beats*.

O *PostgreSQL* precisou ser instalado seguindo sua documentação, maiores informações podem ser encontradas em (GROUP, 1997). Após isso, com auxílio do *PGAdmin4*, um projeto de código aberto que permite desenvolver códigos para banco de dados *PostgreSQL*, um servidor foi criado com o nome *music_server* (veja Figura 8).

Após isso, foi preciso estabelecer uma conexão ao servidor. Para isso, a configuração ilustrada na Figura 9 foi utilizada. No campo *Host name/address* utiliza-se o endereço onde o servidor foi hospedado, que, no caso do trabalho de conclusão de curso, foi o *localhost*, ou seja, um servidor local.

Após a configuração do servidor do banco de dados via *PGAdmin4*, foi necessário configurar a extensão do *pgvector*, seguindo a documentação disponível em (PGVECTOR,

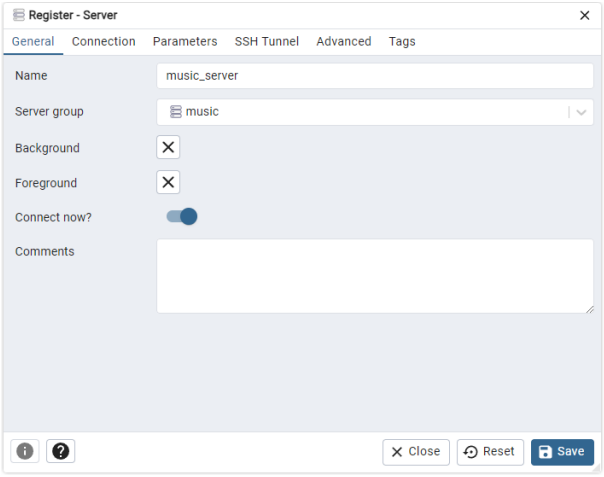

Figura 8 – Definição do nome do servidor. Fonte do autor.

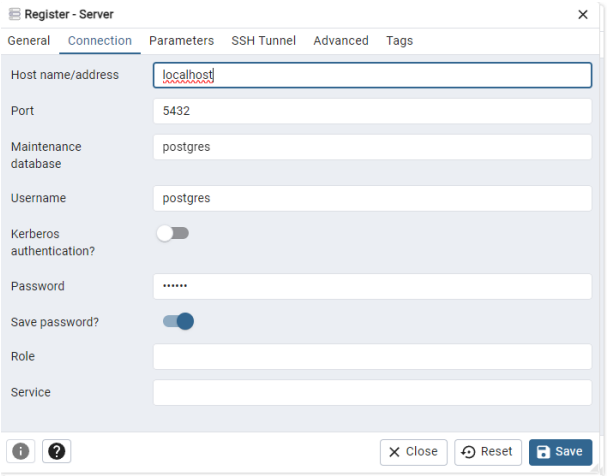

Figura 9 – Definição do nome do servidor. Fonte do autor.

2021). A primeira etapa foi realizar a instalação da extensão, conforme o Algoritmo 2, e, posteriormente, executar uma *query SQL* para a criação de uma tabela, conforme o Algoritmo 3.

Algoritmo 2: Comando SQL para adicionar a extensão de vetor

```
1 CREATE EXTENSION vector;
```

O que resultou na tabela ilustrada na Figura 10, vista pelo *pgAdmin4*.

Também é possível alterar a forma como os vetores são indexados, por padrão, o *pgvector* implementa a busca por vizinhos próximos, porém, se necessário, outras maneiras estão disponíveis, como a HNSW vista na Seção referente a indexação de um SGBDV 2.4.3.2.

Algoritmo 3: Comando SQL para criar a tabela música

```
1 CREATE TABLE musica (
2 id SERIAL NOT NULL PRIMARY KEY,
3 title TEXT NOT NULL,
4 time DECIMAL NOT NULL,
5 musicGenre TEXT,
6 features VECTOR(21) NOT NULL
7 );
```

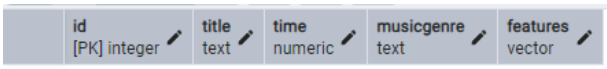

Figura 10 – Tabela gerada via *query SQL*. Fonte do autor.

4.4 Inserção de dados

Utilizou-se a linguagem de programação *Python* com as seguintes bibliotecas: *kagglehub* (KAGGLE, 2024), que facilita realizar o download da base de dados (HAJOORI, 2018a); *psycopg2* (GREGORIO, 2024), biblioteca que permite conectar ao banco de dados PostgreSQL que foi anteriormente criado, e a biblioteca *csv* (PYTHON, 2003), que permite a leitura de arquivos cuja extensão é csv.

Para inserir os dados necessitou-se estabelecer uma conexão com o servidor do banco de dados criado conforme o Algoritmo 4.

Algoritmo 4: Conexão com o banco de dados

```
1 import psycopg2
2 connection ← psycopg2.connect(database = "postgres", user =
    "postgres", password = "123456", host = "localhost", port = 5432)
3 cursor = connection.cursor()
```

Os valores informados na função *psycopg2.connect* foram os valores configurados na Figura 9. O *cursor* é uma classe da biblioteca *psycopg2* que permite executar comandos *SQL*.

Após estar conectado ao servidor, a base de dados utilizada precisa ser extraída. O *Kaggle* (GOLDBLOOM, 2010) juntamente com sua biblioteca para python, *kagglehub* (KAGGLE, 2024), facilita o download e obtenção dos dados desejados, conforme o Algoritmo 5.

Algoritmo 5: Download do arquivo da base de dados

```
1 import kagglehub
2 path ← kagglehub.dataset_download("insiyeah/musicfeatures")
```

Como os dados utilizados foram obtidos da base (HAJOORI, 2018a), o parâmetro informado no método *kagglehub.dataset_download* será: *insiyeah/musicfeatures*, e o retorno dessa chamada, a variável *path*, foi o local onde os dados foram salvos. Nesse exemplo, os dados são dois arquivos do formato csv, porém apenas um deles foi utilizado.

Para finalizar a etapa de inserção de dados, foi preciso ler todas as linhas do arquivo baixado e pegar os dados necessários para serem utilizados no comando de inserção de dados no banco de dados.

Para facilitar, uma variável contendo a query SQL de inserção de dados foi criada conforme o Algoritmo 6.

Algoritmo 6: Query de inserção de dados

```
1 insert_stmt = ("INSERT INTO music (title, time, musicGenre, features) VALUES
    (%s, %s, %s, %s, %s)")
```

Para percorrer todo o arquivo e, para cada linha, com exceção da primeira (que indica o nome de cada coluna da tabela), o Algoritmo 7 foi utilizado para coletar os dados necessários e armazená-los no banco de dados.

Algoritmo 7: Algoritmo de inserção de dados

```
  Input: Arquivo .csv da base de dados
1 import csv
2 csv_reader ← csv.reader(csv_file, delimiter=',')
3 line_count ← 0
4 for row ∈ csv_reader do
5 |   if line_count != 0:
6 |   features ←"["+ row[9]+", "+row[10]+", "+ row[11]+", "+row[12]+",
  |    "+row[13]+ ", "+row[14]+ ", "+row[15]+ ", "+row[16]+ ", "+row[17]+",
  |    "+row[18]+ ", "+row[19]+ ", "+row[20]+ ", "+row[21]+ ", "+row[22]+ ",
  |    "+row[23]+ ", "+row[24]+ ", "+row[25]+ ", "+row[26]+ ", "+row[27]+ ",
  |    "+row[28]+ ", "+row[2]+"]"
7 |   data ← (row[0], row[1], row[29], features)
8 |   cursor.execute(insert_stmt, data)
9 |   line_count += 1
10|   connection.commit()
11 end
```

A variável *csv_reader* é uma matriz, em que cada linha representa uma linha do arquivo lido, e cada coluna um campo da tabela que foi delimitado pelo caracter ", ".

Para armazenar cada música no banco de dados, criou-se uma variável denominada $features$, que armazena todos os 20 valores de MFCC ($row[9] \ldots row[28]$) e o atributo *beats*, representado por $row[2]$. Já o título do áudio era indicado por $row[0]$, o tempo por $row[1]$ e o gênero por $row[29]$. A posição de cada elemento foi determinada pelo arquivo baixado.

Para realmente popular o banco de dados, para cada linha do arquivo foi executado o comando $cursor.execute(insert\_stmt, data)$. Com isso, o comando SQL que havia sido definido na variável $insert\_stmt$ foi executado passando-se as informações definidas na variável $data$.

4.5 Consulta por Similaridade

A extensão *pgvector* já tem implementadas funções para o cálculo de medidas de distância entre vetores mencionadas na Seção 2.3.1. Essas funções foram utilizadas no momento da criação da *query* de seleção, conforme consta na documentação do *pgvector* (GREGORIO, 2024).

Algoritmo 8: Exemplo de query para busca por similaridade

```
1 SELECT * FROM musica ORDER BY features <-> '[3,1,2,4,...,2,4,1]' LIMIT 5;
```

No código apresentado no Algoritmo 8, retornou-se todos os campos da tabela música ordenados pela distância do vetor da linha com o vetor informado no comando, limitado a 5 respostas. O vetor informado para comparação precisa ser do mesmo tamanho que o vetor que foi criado na tabela, que, nesse caso, foi definido um vetor de tamanho 21.

O operador utilizado no comando (nesse exemplo, $< - >$) indica a função de distância que será utilizada ao executar o comando. As opções existentes são: $< - >$, que realiza a comparação pela distância Lp da Equação (2.9), e <=>, que utiliza a função cosseno da Equação (2.10).

Conforme a Figura 7, foram realizadas 3 tipos de consultas: por id, por título e por arquivo de áudio. Para cada uma, foram criados um comando *SQL* e um código em Python para permitir a execução.

Para a implementação em python o Algoritmo 9 foi utilizado. Nele um comando SQL será passado como parâmetro e a resposta será um dicionário contendo o gênero e a quantidade de vezes que o gênero apareceu como resultado. Ele realiza a consulta no banco de dados de músicas semelhantes baseado na query passada como entrada. Ao executar a query via $cursor.execute$ obtém-se o resultado no próprio objeto $cursor$ e, por meio dele, é possível iterar e obter os campos desejados da resposta, no caso do $result[4]$ tem-se o gênero da música.

4.5.1 Consulta pelo id

Para a consulta pelo id utilizando a distância Lp, o comando *SQL* utilizado no Algoritmo 9 foi o Algoritmo 10. Nessa query é informado que será feito a consulta por similaridade do vetor features utilizando a métrica de Lp, indicada pela cláusula $< - >$,

Algoritmo 9: Algoritmo geral para busca no banco de dados

Input: *commando* SQL para a busca da música
Output: Dicionário com os gêneros e a quantidade de vezes que o gênero apareceu na busca

```
1 cursor.execute(commando)
2 for result ∈ cursor do
3 |   print("distance: "+ str(result[0]) + "identifier: "+ str(result[1])+"title: "+
  |    str(result[2])+ "time: "+ str(result[3]) +"musicGenre: "+ str(result[4]))
4 |   dicionario[str(result[4])] = dicionario.setdefault(str(result[4]), 0) + 1
5 end
6 return dicionario
```

	distance double precision	id [PK] integer	title text	time numeric	musicg text	features vector
1	0	4	blues.00012.au	184.5703125	blues	[-207.20808,132....
2	34.63754673118013	56	blues.00023.au	184.5703125	blues	[-227.26477,137....
3	42.50617200128853	275	country.00074.au	135.99917763157896	cou...	[-207.256,130.94...
4	42.818535728736975	250	country.00089.au	143.5546875	cou...	[-205.12329,140....
5	43.191386289550656	43	blues.00016.au	198.76802884615384	blues	[-233.76361,120....
6	44.5088745868977	67	blues.00052.au	151.99908088235293	blues	[-225.26973,117....
7	44.7408390151687 95	86	blues.00088.au	172.265625	blues	[-194.25455,151....
8	45.27038991576061	221	country.00022.au	184.5703125	cou...	[-195.13904,114....
9	46.37693202680347	207	country.00069.au	112.34714673913044	cou...	[-217.14195,136....
10	46.5270156317139	561	jazz.00006.au	135.99917763157896	jazz	[-234.3334,142.7...

Figura 11 – Consulta por id realizada no PGAdmin utilizando Lp. Fonte do Autor.

da seleção da tabela de música cujo id é 4. O resultado dessa consulta realizada pelo programa PGAdmin pode ser visto na Figura 11.

Algoritmo 10: Query para busca por similaridades por id baseado na métrica Lp

```
1 SELECT (features <-> (SELECT features FROM musica WHERE id = 4)) AS
    distance, * FROM musica order by distance LIMIT 10;
```

Já para a distância cosseno mantendo a seleção por id, o comando SQL teve uma pequena alteração em sua cláusula, mudando o $< - >$ por $<=>$, ficando conforme Algoritmo 11. O resultado dessa consulta realizada pelo programa PGAdmin pode ser visto na Figura 12

Algoritmo 11: Query para busca por similaridades por id baseado na métrica cosseno

```
1 SELECT (features <=> (SELECT features FROM musica WHERE id = 4)) AS
    distance, * FROM musica order by distance LIMIT 10;
```

4.5.2 Consulta pelo título

Essa consulta é muito semelhante à consulta por id. A única diferença na implementação foi na cláusula $WHERE$, que ao invés de filtrar pelo id, filtrará pelo título. Para

	distance double precision	id [PK] integer	title text	time numeric	musicgenre text	features vector
1	0	4	blues.00012.au	184.5703125	blues	[-207.20808,132.79918,-15.438986,6
2	0.005491822492284526	56	blues.00023.au	184.5703125	blues	[-227.26477,137.76237,-14.500459,6
3	0.008012254618078973	504	jazz.00062.au	151.99908088235293	jazz	[-168.11578,108.28394,-14.157817,3
4	0.00853786919202637	76	blues.00017.au	172.265625	blues	[-245.75642,140.48593,-2.764772,60
5	0.009542163557689887	496	hiphop.00063.au	92.28515625	hiphop	[-98.80727,71.10884,-10.042009,31.
6	0.010011434424790244	204	country.00033.au	92.28515625	country	[-133.30862,78.986084,-5.6509476,3
7	0.010809831975733597	880	reggae.00050.au	95.703125	reggae	[-104.85737,66.38493,-2.3053617,24
8	0.010916597988888888	43	blues.00016.au	198.76802884615384	blues	[-233.76361,120.67351,-3.171085,39
9	0.01096571052918327	254	country.00087.au	143.5546875	country	[-170.87102,124.80758,-19.249126,3
10	0.010976944766617147	883	reggae.00077.au	143.5546875	reggae	[-151.75853,105.12133,-6.1730165,4

Figura 12 – Consulta por id realizada no PGAdmin utilizando cosseno. Fonte do Autor.

	distance double precision	id [PK] integer	title text	time numeric	musicgenre text	features vector
1	0	338	disco.00091.au	99.38401442307692	disco	[-116.94722,98.49051,-19
2	29.34474422570207	938	rock.00000.au	123.046875	rock	[-116.60803,109.26915,-2
3	29.42475112900543	414	hiphop.00083.au	103.359375	hiphop	[-121.99201,89.59093,-25
4	30.513310764141132	941	rock.00002.au	107.666015625	rock	[-120.4687,117.58884,-27
5	32.006621629359756	235	country.00086.au	117.45383522727273	country	[-122.42455,111.474556,-
6	32.700472823058114	954	rock.00077.au	107.666015625	rock	[-103.9137,101.89537,-10
7	33.433359848828390	452	hiphop.00082.au	107.666015625	hiphop	[-122.766556,95.037415,-
8	33.94770291497092	398	disco.00060.au	129.19921875	disco	[-121.073105,116.9522,-2
9	34.177174014725274	386	disco.00062.au	107.666015625	disco	[-107.85015,81.94442,-30
10	34.33477227118395	278	country.00088.au	103.359375	country	[-121.92424,123.23469,-8

Figura 13 – Consulta por título realizada no PGAdmin utilizando cosseno. Fonte do Autor.

isso, basta utilizar o Algoritmo 12 no Algoritmo 9. O resultado dessa consulta realizada pelo programa PGAdmin pode ser visto na Figura 13.

Algoritmo 12: Query para busca por similaridades por título baseado na métrica Lp

```
1 SELECT (features <-> (SELECT features FROM musica WHERE title =
    'disco.00091.au')) AS distance, * FROM musica order by distance LIMIT 10;
```

Como a cláusula para comparação da distância foi a $<->$, no Algoritmo 12, então a métrica utilizada é a Lp, análogo ao realizado no Algoritmo 11 é possível realizar no Algoritmo 12 para que a métrica de distância seja a cosseno, como ilustrado na Figura 13.

4.5.3 Consulta por arquivo de áudio

Essa forma de consulta necessitou de mais manipulações. Nela, foi necessário realizar etapas de extração de atributos para permitir a comparação por similaridade.

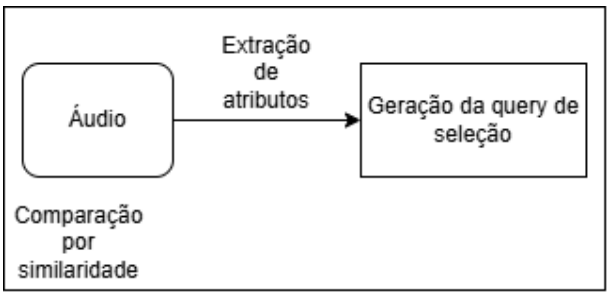

Figura 14 – Consulta por arquivo de áudio. Fonte do autor.

Conforme demonstrado na Figura 14, a entrada será um arquivo de áudio que será lido pela biblioteca *librosa*, e será construído seu vetor de atributos compostos pelos 20 valores de MFCC e a extração do *beats*. Esse vetor precisou ser gerado igual aos dados que já foram populados.

A primeira etapa realiza a extração do vetor atributo que será utilizado no comando *SQL* para consulta por similaridade por áudio, esse algoritmo é análogo ao Algoritmo 1, a única diferença está no retorno, que será um vetor cujos primeiros valores são a média de cada vetor MFCC e o último valor será a média do beats. O comando SQL para comparação por similaridade é semelhante aos demais, a única diferença está no vetor que é passado para a cláusula $< - >$ ou $<=>$, que será o vetor que foi extraído no Algoritmo 1, conforme visto no Algoritmo 13.

Abordagens demonstradas nas Seções 4.1 e 4.4 foram utilizadas para a função de extração de atributos e geração do vetor de atributos. O código para essa implementação foi:

Algoritmo 13: Query para busca por similaridades informando um vetor de atributos

Input: Vetor de atributos p

```
1 SELECT (features <-> p) AS distance, * FROM musica order by distance LIMIT
    10;
```

4.6 Considerações Finais

Ao final dessa etapa do projeto, um banco de dados foi populado com os dados fornecidos pela base (HAJOORI, 2018a). Além disso, funções em *Python* que utilizam o comando *SQL* com a extensão do *pgvector* para realizar a comparação por similaridade foram definidas para permitir realizar consultas por id ou título ou arquivo de música, resultando em um ranking limitado por um valor passado como parâmetro na função. Além disso, foram criadas duas formas de realizar o cálculo de distância que serão avaliadas no Capítulo 5.

5 Resultados

Para a análise dos resultados, consultas no banco de dados populados com a base de dados (HAJOORI, 2018a) foram realizadas por meio dos Algoritmos descritos no Capítulo 4. A avaliação será referente aos gêneros musicais retornados pelas consultas de similaridade de músicas, semelhante ao feito em (KNEES; SCHEDL, 2013).

Nessa avaliação, realiza-se uma busca por similaridade de uma música e contabiliza-se a quantidade de músicas do mesmo gênero que foram retornadas. Para a consulta, utilizou-se os Algoritmos 9, 10 e 11. Os resultados das consultas foram submetidos às taxas de precisão, taxa de *recall* e *F-measure*, conforme descritos na Seção 2.3.2. Para isso, definiu-se como "resultados relevantes" as músicas de mesmo gênero da música utilizada. Também limitou-se o número de retornos a 100 músicas, isto é, o algoritmo retornou as 100 músicas mais similares à música informada como entrada.

Na base de dados utilizada, (HAJOORI, 2018a), havia 10 gêneros musicais: *reggae*, clássico, *hip hop*, *disco*, *jazz*, *pop*, *country*, *blues*, *rock* e *metal*. Para cada gênero, 100 músicas estavam disponíveis, fornecendo assim uma base de dados de 1000 arquivos.

Observou-se que a eficiência do algoritmo pode variar bastante dentro de um mesmo gênero. Por exemplo, a música na posição 748 do banco de dados é do gênero *pop*. O algoritmo retorna, na distância Lp, 3 músicas do gênero *pop*, 21 *blues*, 16 *reggae*, 19 clássicas, 5 *country*, 6 *rock*, 2 *hip hop*, 27 *jazz* e 1 *disco*, fornecendo uma taxa de precisão de 0.03. Em contrapartida, a música na posição 708 do banco de dados também é do gênero *pop*, e seu retorno, na distância Lp, consiste em 60 músicas do gênero *pop*, 3 *reggae*, 4 *country*, 3 *rock*, 10 *hip hop*, 2 *jazz* e 18 *disco*, fornecendo uma taxa de precisão de 0.6. Isso indica que, mesmo que uma música seja "classificada" em um único gênero musical, ela pode apresentar características que a "aproxime" a músicas de diversos outros estilos.

Além disso, a eficiência do algoritmo também é bem distinta de acordo com os gêneros musicais. Para se chegar a essa conclusão, utilizou-se o retorno do Algoritmo 9 com o comando SQL do Algoritmo 10 que é um dicionário que contém a quantidade de cada música por gênero, e, com isso, foi possível calcular as taxas de precisão e recall. Ao final do processo, definiu-se "taxa" como sendo a média simples das taxas de precisão.

Uma vez que o banco de dados possui 100 músicas de cada gênero e exigiu-se que a quantidade de músicas retornadas seja 100, pode-se ver (comparando-se as Equações (2.11), (2.12) e (2.13)) que as taxas de precisão, *recall* e *F-measure* são iguais. É por esse motivo que somente a taxa de precisão foi considerada nessa análise. A Tabela 2 apresenta os resultados obtidos.

Gênero	Taxa (distância Lp)	Taxa (distância cosseno)
Blues	0.17110000000000014	0.1598
Clássico	0.46289999999999987	0.5088999999999999
Country	0.17550000000000002	0.19450000000000003
disco	0.2087	0.19069999999999993
hip hop	0.17679999999999996	0.14640000000000003
jazz	0.2112	0.23630000000000007
metal	0.34420000000000006	0.3518
pop	0.4114000000000001	0.3378999999999999
reggae	0.23420000000000002	0.24860000000000004
rock	0.16129999999999992	0.16140000000000007

Tabela 2 – Eficiência do algoritmo conforme os gêneros musicais.

Isso significa que alguns gêneros musicais são bem similares a outros do banco de dados (e por isso suas músicas se aproximam de músicas de outros gêneros), enquanto outros (como o clássico e o pop) são, de certa forma, bem distantes de outros gêneros musicais disponíveis no banco de dados.

Os dados da Tabela 2, que podem ser melhor visualizados no gráfico da Figura 15, também permitem concluir que a escolha da melhor distância a ser utilizada no algoritmo varia conforme o gênero musical. Para o gênero *blues*, por exemplo, a taxa de precisão média foi maior utilizando-se a distância Lp, enquanto que no gênero clássico a taxa de precisão média foi maior com a distância cosseno.

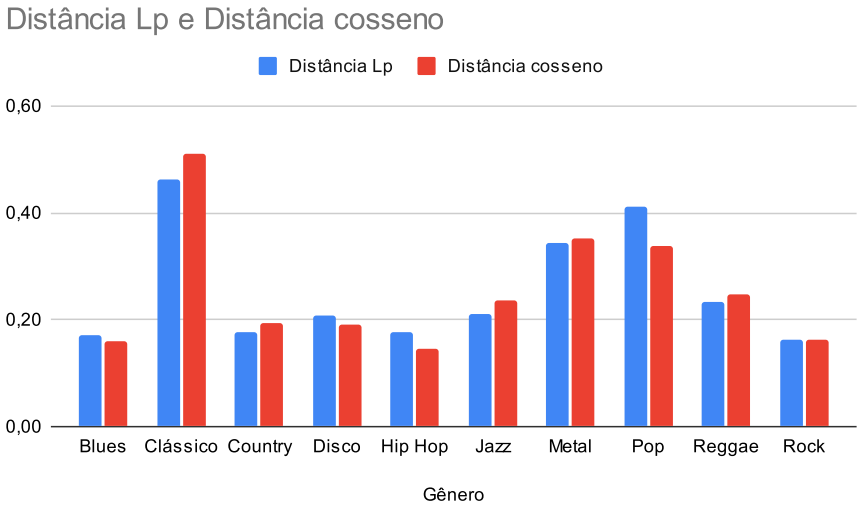

Figura 15 – Taxa de eficiência do algoritmo conforme os gêneros musicais.

O algoritmo criado para realizar as consultas descritas nesse trabalho de conclusão de curso está disponível no seguinte repositório do Github: (SOUSA, 2025).

6 Conclusão

Com base nos resultados analisados no Capítulo 5, nota-se que é possível representar um dado complexo (no caso deste trabalho de conclusão de curso, uma música) por meio de um vetor de atributos em um banco de dados relacional vetorial. De fato, a extensão *pgvector* para o *PostgreSQL* atendeu bem ao propósito de armazenar um vetor de atributos e realizar a comparação por distância Lp e distância cosseno.

As taxas de avaliação descritas no Capítulo 5 não foram tão boas. Percebe-se que existem gêneros musicais para os quais as taxas são baixas, enquanto para outros obtém-se resultados mais satisfatórios. Uma possível forma de se melhorar os resultados seria a utilização de outros atributos para a representação das músicas, que estejam mais próximos à maneira como o algoritmo foi validado.

Além disso, a base de dados utilizada neste trabalho não permitia a determinação das músicas usadas para sua obtenção. Portanto, uma forma de incrementar os resultados deste trabalho seria utilizar outra base de dados, como base de dados fornecidas pela API do Spotify, para se avaliar manualmente os resultados fornecidos pelo algoritmo. Isso permite uma avaliação de quais atributos podem ser modificados, para que se altere a distância entre as músicas e, assim, obtenha-se um algoritmo mais eficiente conforme a sua validação.

Outro possível incremento para este trabalho seria a utilização de outras fórmulas de distância entre vetores, e adaptar a extensão do *pgvector* para permitir a consulta por meio dessas novas métricas. Além disso, outras maneiras de representar a música podem ser exploradas, como o uso de LLMs para melhorar na comparação.

Outros possíveis estudos seria quanto as medidas de avaliação como o *Hit em K* e *MMR em K*, que avalia os K vetores mais próximos. De modo geral, o objetivo do trabalho de conclusão de curso foi obtido: a representação de músicas em um banco de dados relacional vetorial e, por meio de comandos *SQL*, obter um *ranking* das músicas semelhantes a uma música fornecida como entrada

Referências

ANDRADE, M. d. F. *Avaliação do desempenho de sistemas armazém de dados baseados em NoSQL*. Dissertação (Trabalho de Conclusão de Curso) — Universidade Federal de Uberlândia, 2024. Citado 3 vezes nas páginas 28, 33 e 34.

BARIONI, M. C. N. *Operações de consulta por similaridade em grandes bases de dados complexos*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação: Universidade de São Paulo, 2006. Citado 3 vezes nas páginas 17, 26 e 33.

BELSKY, I. T. I. *Powerful Digital Assets & Tools for Video Creators | Artlist — artlist.io*. 2015. <https://artlist.io/>. [Accessed 25-04-2025]. Citado na página 36.

BISPO, L. L. d. S. *Uso de metadados e compressão de áudio digital em plataformas de serviço streaming*. Dissertação (Trabalho de Conclusão de Curso) — Faculdade de Ciência da Informação, Universidade de Brasília Brasília (DF), 2016. Citado 2 vezes nas páginas 12 e 21.

BORJIAN, N. Query-by-example music information retrieval by score-based genre prediction and similarity measure. *International Journal of Multimedia Information Retrieval*, Springer, v. 6, p. 155–166, 2017. Citado na página 32.

DERUTY, E. *Intuitive understanding of MFCCs — derutycsl*. 2022. <https://medium.com/@derutycsl/intuitive-understanding-of-mfccs-836d36a1f779>. [Accessed 22-04-2025]. Citado 2 vezes nas páginas 11 e 25.

DINIZ, P.; SILVA, E. da; NETTO, S. *Processamento Digital de Sinais - 2.ed.: Projeto e Análise de Sistemas*. Bookman Editora, 2014. ISBN 9788582601242. Disponível em: <https://books.google.com.br/books?id=HoWaAgAAQBAJ>. Citado na página 22.

DOUZE, M. et al. *The Faiss library*. 2025. Disponível em: <https://arxiv.org/abs/2401.08281>. Citado 2 vezes nas páginas 17 e 33.

FACHINI, A. R.; HEINEN, M. R. Aplicação de mfcc para modelar sons de instrumentos musicais. In: *11th proceedings of brazilian congress on computational intelligence*. [S.l.: s.n.], 2016. p. 31. Citado na página 32.

GOLDBLOOM, A. *Kaggle: Your Machine Learning and Data Science Community — kaggle.com*. 2010. <https://www.kaggle.com/>. [Accessed 25-04-2025]. Citado 2 vezes nas páginas 36 e 40.

GREGORIO, F. D. *psycopg2 — pypi.org*. 2024. <https://pypi.org/project/psycopg2/>. [Accessed 25-04-2025]. Citado 2 vezes nas páginas 40 e 42.

GROUP, P. G. D. *PostgreSQL — postgresql.org*. 1997. <https://www.postgresql.org>. [Accessed 25-04-2025]. Citado na página 38.

HAJOORI, I. *Music features — kaggle.com*. 2018. <https://www.kaggle.com/datasets/insiyeah/musicfeatures>. [Accessed 25-04-2025]. Citado 6 vezes nas páginas 36, 38, 40, 41, 45 e 46.

HAJOORI, I. *Music-Tagging/dataset_extraction.ipynb at master · Insiyaa/Music-Tagging — github.com*. 2018. <https://github.com/Insiyaa/Music-Tagging/blob/master/dataset_extraction.ipynb>. [Accessed 25-04-2025]. Citado na página 37.

HELERBROCK, R. *Som: o que é, características, fenômenos envolvidos - Mundo Educação — mundoeducacao.uol.com.br*. 2019. <https://mundoeducacao.uol.com.br/fisica/o-que-som.htm>. [Accessed 22-04-2025]. Citado 2 vezes nas páginas 11 e 19.

HURLEY, S. C. e. J. K. C. *YouTube — youtube.com*. 2005. <https://www.youtube.com/>. [Accessed 25-04-2025]. Citado na página 36.

KAGGLE. *GitHub - Kaggle/kagglehub: Python library to access Kaggle resources — github.com*. 2024. <https://github.com/Kaggle/kagglehub>. [Accessed 25-04-2025]. Citado na página 40.

KAMUNI, N. et al. Advancing Audio Fingerprinting Accuracy with AI and ML: Addressing Background Noise and Distortion Challenges . In: *2024 IEEE 18th International Conference on Semantic Computing (ICSC)*. Los Alamitos, CA, USA: IEEE Computer Society, 2024. p. 341–345. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/ICSC59802.2024.00064>. Citado na página 17.

KNEES, P.; SCHEDL, M. Music similarity and retrieval. In: *proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. [S.l.: s.n.], 2013. p. 1125–1125. Citado 4 vezes nas páginas 22, 26, 27 e 46.

LAUREANO, M. et al. *Reconhecimento e Análise Musical de Instrumentos Melódicos com Redes Neurais*. Dissertação (Trabalho de Conclusão de Curso) — Universidade Federal de Santa Catarina, 2022. Citado na página 32.

LTD, R. S. *DB-Engines Ranking — db-engines.com*. 2013. <https://db-engines.com/en/ranking>. [Accessed 29-04-2025]. Citado na página 29.

MALKOV, Y. A.; YASHUNIN, D. A. *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs*. 2018. Disponível em: <https://arxiv.org/abs/1603.09320>. Citado na página 33.

MCFEE, B. et al. *librosa/librosa: 0.11.0*. Zenodo, 2025. Disponível em: <https://doi.org/10.5281/zenodo.15006942>. Citado na página 37.

MED, B. *Teoria da música*. MusiMed, 1996. ISBN 9788585886028. Disponível em: <https://books.google.com.br/books?id=G2lWAAAACAAJ>. Citado na página 19.

MúSICA, D. a. *Timbre | Descomplicando a Música — descomplicandoamusica.com*. 2023. <https://www.descomplicandoamusica.com/timbre/>. [Accessed 22-04-2025]. Citado 2 vezes nas páginas 11 e 20.

NAM, J. *Learning feature representations for music classification*. Tese (Doutorado) — Stanford University, 2012. Citado na página 32.

OLIPHANT, T. *NumPy — numpy.org*. 2005. <https://numpy.org/>. [Accessed 25-04-2025]. Citado na página 37.

PAN, J. J.; WANG, J.; LI, G. Survey of vector database management systems. *arXiv e-prints*, p. arXiv–2310, 2023. Citado na página 29.

PGVECTOR. *GitHub - pgvector/pgvector: Open-source vector similarity search for Postgres — github.com*. 2021. <https://github.com/pgvector/pgvector>. [Accessed 16-04-2025]. Citado na página 39.

PYTHON. *csv — CSV File Reading and Writing — docs.python.org*. 2003. <https://docs.python.org/3/library/csv.html>. [Accessed 25-04-2025]. Citado na página 40.

RAMAKRISHNAN, R.; GEHRKE, J. *Database management systems third edition*. [S.l.]: McGraw-Hill Education, 2003. Citado 2 vezes nas páginas 28 e 33.

RAZLOGOVA, E. Shazam: The blind spots of algorithmic music recognition and recommendation. In: _____. [S.l.]: University of Michigan Press, 2018. p. 257–266. Citado na página 32.

RODRIGUES, W. B. *NoSQL: a análise da modelagem e consistência dos dados na era do Big Data*. Dissertação (Mestrado) — Pontifícia Universidade Católica de São Paulo, 2017. Citado na página 28.

SHUVAEV, S.; GIAFFAR, H.; KOULAKOV, A. A. *Representations of Sound in Deep Learning of Audio Features from Music*. 2017. Disponível em: <https://arxiv.org/abs/1712.02898>. Citado na página 32.

SOUSA, V. G. *Music extraction — gist.github.com*. 2025. <https://gist.github.com/ViniciusGuardieiroSousa/4d5052c7c43d52f5f429b11f818ae936>. [Accessed 27-05-2025]. Citado na página 47.

TAIEF, A. M. *Application of LLMs and Embeddings in Music Recommendation Systems*. Dissertação (Mestrado) — UiT Norges arktiske universitet, 2024. Citado na página 33.

TAIPALUS, T. Vector database management systems: Fundamental concepts, use-cases, and current challenges. *Cognitive Systems Research*, v. 85, p. 101216, 2024. ISSN 1389-0417. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389041724000093>. Citado 4 vezes nas páginas 11, 29, 33 e 35.

TAMBOLI, A. I.; KOKATE, R. D. Query based relevant music genre retrieval using adaptive artificial neural network for multimedia applications. *Multimedia Tools and Applications*, Springer, v. 81, n. 22, p. 31603–31629, 2022. Citado na página 17.

TAMBOLI, A. I.; KOKATE, R. D. Query based relevant music genre retrieval using adaptive artificial neural network for multimedia applications. *Multimedia Tools and Applications*, Springer, v. 81, n. 22, p. 31603–31629, 2022. Citado na página 32.

TANENBAUM, A. S.; BOS, H. *Sistemas operacionais modernos*. [S.l.]: Bookman Editora, 2024. Citado na página 21.

TRAN, T.; SWEENEY, R.; LEE, K. Adversarial mahalanobis distance-based attentive song recommender for automatic playlist continuation. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2019. (SIGIR'19), p. 245–254. ISBN 9781450361729. Disponível em: <https://doi.org/10.1145/3331184.3331234>. Citado na página 17.

TZANETAKIS, G. *Manipulation, Analysis and retrieval systems for audio signals*. Tese (Doutorado) — Princeton University, 2002. Citado na página 25.

WANG, A. L. An industrial-strength audio search algorithm. In: *ISMIR 2003, 4th Symposium Conference on Music Information Retrieval*. [S.l.: s.n.], 2003. p. 7–13. In , S. Choudhury and S. Manus, Eds., The International Society for Music Information Retrieval. http://www.ismir.net: ISMIR, October , pp. . [Online]. Available: http://www.ee.columbia.edu/ dpwe/papers/Wang03-shazam.pdf. Citado na página 17.

WANG, J. et al. Milvus: A purpose-built vector data management system. In: *Proceedings of the 2021 International Conference on Management of Data*. [S.l.: s.n.], 2021. p. 2614–2627. Citado 2 vezes nas páginas 17 e 33.

YUN, S.; LIM, Y.-k. User experience with llm-powered conversational recommendation systems: A case of music recommendation. *arXiv preprint arXiv:2502.15229*, 2025. Citado na página 33.