FACULTAD DE COMPUTACIÓN

Base de Datos 2

Base de Datos Multimodal para Recuperación por Contenido

Estudiante	**Participación (%)**
Jimena Mamani Ramirez	100 %

Profesor:
Heider Sanchez

Diciembre 2025

Contents

1. Introducción

Este proyecto aborda el diseño e implementación de una base de datos multimodal enfocada en la recuperación de información textual y de audio. Dicha base permite la búsqueda eficiente por contenido a través de texto y características acústicas (Bag of Audio Words).

Se justifica esta necesidad ante el crecimiento exponencial de datos no estructurados y la demanda de sistemas que integren múltiples modalidades para mejorar la calidad de las respuestas.

2. Backend – Índice Invertido para Texto

2.1 Construcción del Índice: SPIMI

Para el índice invertido textual, se implementó una versión adaptada del algoritmo **SPIMI (Single-Pass In-Memory Indexing)** en el archivo `spimi.py`. Este algoritmo permite la construcción eficiente de índices sobre colecciones grandes, operando directamente sobre bloques en memoria y escribiendo los resultados en disco conforme se llena la RAM.

En nuestra implementación, cada documento es preprocesado mediante tokenización, normalización y eliminación de ruido textual (ver `preprocessor.py`). Luego, se calcula la frecuencia de término (TF) y se utiliza una fórmula de TF-IDF simplificada en línea, donde el IDF se calcula con base en la cantidad de documentos vistos hasta el momento:

$$\text{idf}(t) = \log\left(\frac{N}{1 + df(t)}\right)$$

Las listas de postings almacenan tuplas del tipo (doc_id, peso), y se guarda también la norma del vector TF-IDF por documento para facilitar la normalización durante la recuperación. Finalmente, todo el índice se guarda en formato JSON en disco mediante la función interna `_save_index`.

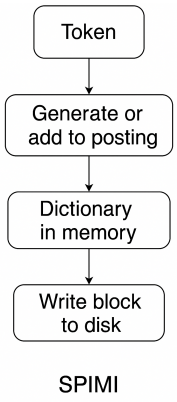

Figure 1: Esquema de funcionamiento del algoritmo SPIMI

2.2 Recuperación y Ranking

La recuperación se implementa en `search.py` dentro de la clase `SPIMISearcher`. A partir de una consulta textual, se aplica el mismo preprocesamiento y se calcula un vector TF-IDF para la consulta. Este vector se compara con los documentos indexados usando la **Similitud de Coseno**, cuyo cálculo se realiza como:

$$\text{sim}(q, d) = \frac{\sum_{t \in q \cap d} w_{q,t} w_{d,t}}{\|q\| \cdot \|d\|}$$

Donde los pesos se calculan como $w_{q,t} = tf \cdot idf$. El sistema retorna los documentos con mayor puntuación normalizada. Se emplea una estructura tipo diccionario para acumular los puntajes parciales y luego se aplica ordenamiento descendente. Esta lógica permite soportar eficientemente consultas top-k.

3. Backend – Índice Invertido para Descriptores Locales (Audio)

3.1 Descripción del Pipeline

Para los archivos de audio, se desarrolló un sistema de representación basado en **Bag of Audio Words (BoAW)**. Este pipeline consta de varias etapas encadenadas:

- Primero, se extraen **coeficientes MFCC** por frame mediante la librería `librosa`. Esto genera una matriz de características por cada audio.
- Se recolectan todos los vectores MFCC del conjunto de entrenamiento y se agrupan usando **KMeans**, generando un vocabulario acústico o diccionario.
- Cada archivo de audio se representa como un vector BoAW (histograma de frecuencias de palabras acústicas), el cual puede ser ponderado opcionalmente con TF-IDF.

Este procesamiento se encuentra implementado en el módulo `audio_indexer.py` (extracción de MFCCs y vectorización e indexación de histogramas).

3.2 Búsqueda por Similitud

La recuperación se realiza comparando el vector BoAW del audio de consulta con los vectores almacenados. Las métricas de similitud empleadas son la **distancia de coseno** y la **distancia euclidiana**.

El sistema permite dos modos de operación:

- **Secuencial**: se compara directamente contra todos los histogramas usando numpy o scipy.
- **Indexado**: en desarrollo, se planea usar un índice invertido acústico para acelerar las búsquedas.

Toda la lógica de búsqueda está en `audio_indexer.py` y se complementa con herramientas de evaluación top-k para medir precisión y tiempos de respuesta.

3.3 Análisis de Eficiencia y Maldición de la Dimensionalidad

- **Problema:** MFCCs y BoAW generan vectores de alta dimensión.
- **Impacto:** Reducción de rendimiento en búsquedas KNN.

3.4 Integración con Backend y Frontend

El índice BoAW fue integrado al backend desarrollado en `FastAPI` mediante el endpoint `/search_audio`. Este recibe un archivo `.wav` o `.mp3` y ejecuta el pipeline descrito: extracción de MFCCs, vectorización y comparación contra los histogramas almacenados.

Adicionalmente, se implementó un sistema de **logging automático** para registrar los tiempos de respuesta de cada método (secuencial e indexado) en el archivo `logs_multimedia.csv`, permitiendo trazabilidad y evaluación posterior. Las columnas registradas son:

- `metodo`: tipo de búsqueda (secuencial o indexado).

- `tiempo_respuesta`: tiempo total en milisegundos.

En el frontend `Streamlit`, se rediseñó la sección `Buscar por Audio` para incluir:

- Visualización del espectrograma o forma de onda del audio cargado.
- Reproducción interactiva del audio.
- Visualización comparativa de los resultados obtenidos por los dos métodos de búsqueda.
- Despliegue del tiempo de respuesta de cada método junto a sus resultados top-k.

3.5 Avance en el Índice Invertido Acústico

Se realizó la implementación de un **índice invertido para descriptores acústicos**, inspirado en el enfoque textual SPIMI, adaptado al dominio de audio. En lugar de palabras, se indexan **identificadores de centroides de KMeans**, que representan palabras acústicas.

El índice almacena:

- Por cada palabra acústica (centroide): una lista de documentos (archivos de audio) en los que aparece.
- Las frecuencias relativas o TF de cada palabra por documento.

Este índice busca permitir recuperación eficiente mediante **inversión BoAW-to-document**, acelerando consultas sin necesidad de comparar directamente contra todos los vectores.

3.6 Comparación de Métodos y Métricas de Evaluación

Para evaluar los métodos de búsqueda implementados, se utiliza una métrica de **precisión top-k**, midiendo cuántos de los resultados recuperados coinciden con el audio de referencia (según sus metadatos).

Además, los tiempos de respuesta permiten comparar escalabilidad:

- El método secuencial tiene un costo lineal con respecto al número de documentos y la dimensión del vector.
- El índice invertido busca mejorar esto al reducir la cantidad de histogramas contra los que se compara.

4. Frontend

El frontend del sistema se desarrolló con **Streamlit**, permitiendo una interacción amigable con el usuario. Se estructura en tres módulos principales:

4.1 Consulta SQL

La primera sección permite realizar consultas semánticas sobre metadatos musicales usando sintaxis SQL. La entrada del usuario es interpretada mediante un parser que utiliza la librería `mo-sql-parsing`, que traduce la sintaxis SQL a un formato estructurado (diccionario) que luego se interpreta para consultar sobre archivos CSV indexados previamente. El frontend se conecta al backend mediante una API REST.

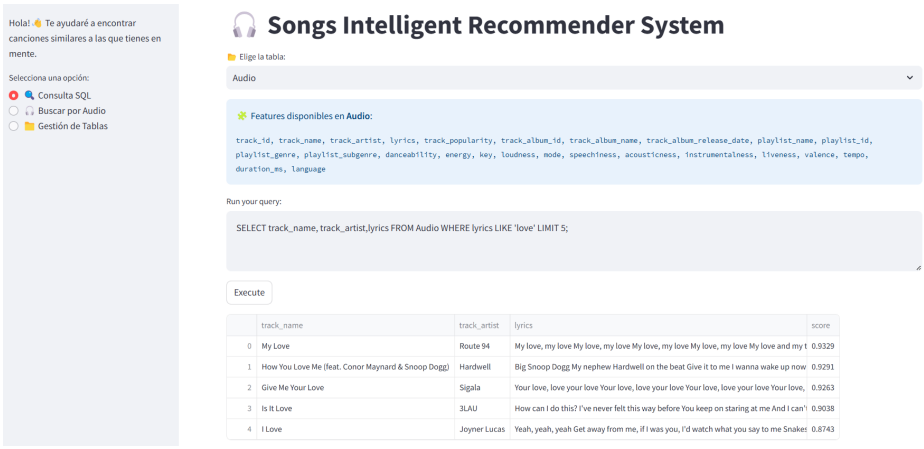

Figure 2: Interfaz de consulta SQL

4.2 Búsqueda por Audio

El segundo módulo permite al usuario cargar un archivo de audio (.mp3 o .wav). Este archivo es enviado al backend, donde se extraen sus características (MFCCs), se transforma en BoAW y se compara con los vectores indexados previamente.

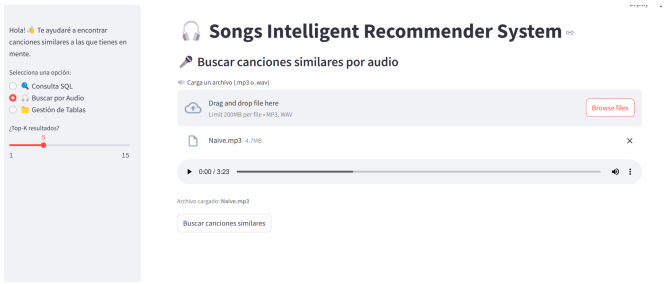

Figure 3: Interfaz para búsqueda por similitud acústica

4.3 Gestión de Tablas

La sección final del frontend permite crear nuevas tablas (carpetas) y cargar archivos CSV, los cuales son procesados e indexados automáticamente. Esto permite al usuario administrar múltiples colecciones de datos.

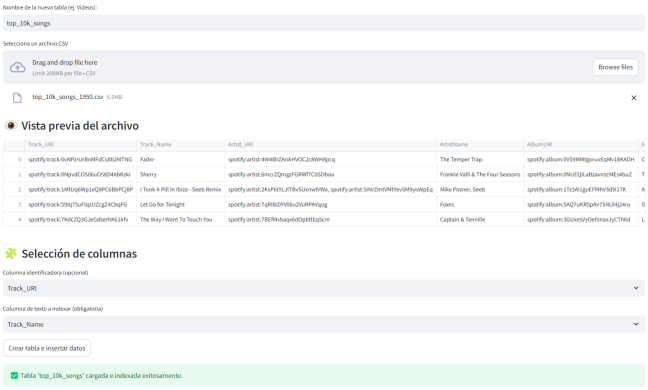

Figure 4: Interfaz para gestión de tablas y carga de CSV

5. Experimentación

5.1 Análisis Exploratorio

5.1.1 Tiempos de respuesta por consulta

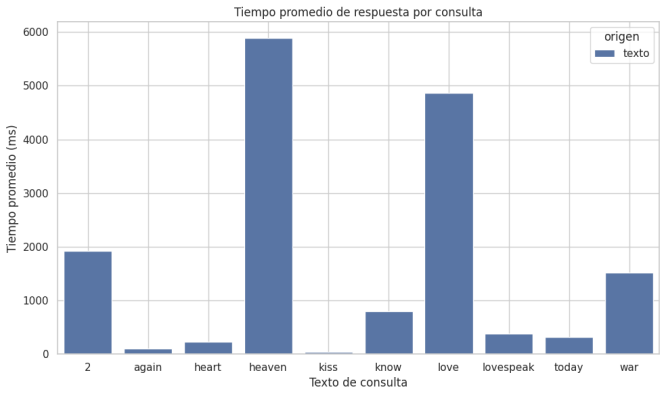

Figure 5: Gráfica de comparación de tiempo de respuesta por consulta

Se observa que los tiempos de respuesta varían significativamente entre consultas, destacándose algunas palabras con tiempos medios mucho mayores que otras, lo que sugiere diferencias importantes en la complejidad o selectividad de las búsquedas.

5.1.2 Tiempos de respuesta por método utilizado

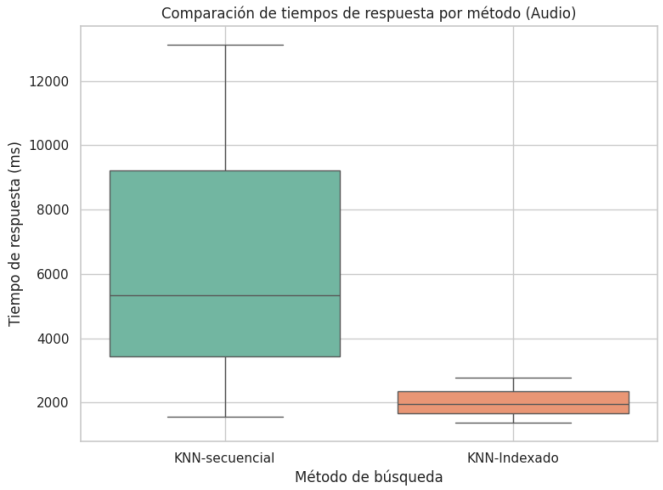

Figure 6: Gráfica de comparación de tiempo de respuesta por método

En la gráfica, el método KNN indexado presenta tiempos de respuesta considerablemente menores y con menor dispersión en comparación con el KNN secuencial, evidenciando una mejora clara en eficiencia al utilizar índices.

5.2 Comparación con PostgreSQL

Se comparó el rendimiento de la implementación actual del índice invertido con los obtenidos por índices utilizando PostgreSQL.

A continuación se puede visualizar la tabla de los resultados de los tiempos de respuesta:

Table 1: Tiempos de respuesta (ms) para distintos tamaños de N

N	**MyIndex**	**PostgreSQL**
N = 1000	23.13	1.437
N = 3000	25.77	0.544
N = 10000	126.26	3.441
N = 32000	339.79	3.606
N = 65000	694.05	1.639

Los resultados de la tabla muestran que, a medida que aumenta N, el tiempo de respuesta de MyIndex SPIMI crece de forma casi lineal o incluso superlineal (de 23.13 ms a 694.05 ms), mientras que PostgreSQL se mantiene siempre por debajo de 4 ms e incluso presenta valores muy bajos en varios tamaños (0.544–3.606 ms).

Se usó el motor de búsqueda de texto completo de PostgreSQL: se convierte el texto a un tsvector (tokens ya normalizados) y crea un índice GIN sobre esa columna.

Las consultas usan operadores @@ (match entre tsvector y tsquery) y funciones de ranking como ts_rank_cd.

Así se logró comparar el índice invertido SPIMI (MyIndex) contra un índice invertido real de un SGBD: mismos documentos, misma consulta, comparando tiempo de respuesta y calidad/orden del ranking.

Table 2: Tiempos de respuesta (ms) para distintos tamaños de N

N	**KNN-Secuencial**	**KNN-Indexado**
1	13126.13	2780.25
2	5332.42	1950.08
3	1549.22	1388.15
4	21279.32	4216.42

*$K = 8$.

La tabla muestra que, para todos los valores de N, el KNN-Indexado tiene tiempos de respuesta notablemente menores que el KNN-Secuencial (por ejemplo, 13126.13 ms vs. 2780.25 ms en la primera fila). Esto indica una mejora consistente de varios múltiplos en el tiempo de búsqueda cuando se utiliza el índice.

Además, aunque los tiempos fluctúan entre ejecuciones, el método secuencial presenta valores muy altos y más dispersos (como 21279.32 ms en la cuarta fila), mientras que el método indexado se mantiene en un rango más acotado (máximo alrededor de 4216.42 ms). En conjunto, la evidencia de la tabla respalda que el uso de un índice para KNN reduce significativamente el costo computacional por consulta y hace el comportamiento temporal más estable a través de distintas pruebas.

6. Conclusiones

Se construyó una plataforma multimodal capaz de procesar tanto texto como audio, con un backend eficiente basado en índices invertidos y un frontend usable e interactivo. El sistema es extensible, modular y fácilmente evaluable mediante experimentos controlados.

Los resultados muestran que, conforme aumenta N, los tiempos de respuesta de la implementación manual crecen de manera marcada, pasando de decenas de milisegundos a varios cientos, mientras que PostgreSQL se mantiene siempre en el rango de milisegundos bajos para todos los tamaños evaluados. Esto indica que el rendimiento de la implementación manual es mucho más sensible al crecimiento del conjunto de datos que el del motor de base de datos.

PostgreSQL logra un mejor desempeño porque utiliza búsqueda de texto completo sobre una columna ‘tsvector‘ con un índice GIN, que es una estructura de datos diseñada específicamente para indexar colecciones de valores múltiples (como términos de texto) y responder consultas de pertenencia de forma muy eficiente. Al almacenar tokens ya normalizados en ‘tsvector‘ y apoyarse en el índice GIN, PostgreSQL reduce el trabajo necesario en tiempo de consulta (menos comparaciones y menos lecturas de páginas), lo que se traduce en tiempos de respuesta más bajos y que escalan mejor con el tamaño de los datos que la implementación manual.

7. Anexos

7.1 Repositorio del proyecto

En el siguiente link se puede revisar el desarrollo completo del proyecto: **Proyecto 2 - BD Multimodal**

8. Referencias Bibliográficas

1. Klahnakoski, M. (s.f.). *mo-sql-parsing (v7.1.0)* [Repositorio GitHub]. Recuperado el 14 de julio de 2025, de `https://github.com/klahnakoski/mo-sql-parsing`

2. Rabiner, L., Schafer, R. (2011). *Introduction to digital speech processing.* Foundations and Trends in Signal Processing, 1(1–2), 1–194. https://doi.org/10.1561/2000000001

3. Zobel, J., Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys*, 38(2), 6. https://doi.org/10.1145/1132956.1132959