República Bolivariana de Venezuela

Universidad Nacional Experimental de Guayana

Ingeniería informática

Asignatura: BASE DE DATOS II

Sección: 1

PROYECTO Nº 1

BASES DE DATOS BASADAS EN COLUMNAS

APACHE CASSANDRA

Docente:

CLINIA CORDERO

Integrantes:

ARIADNA AVILA

KEINER FRONTADO

SANTIAGO SANCHEZ

Puerto Ordaz 06 de junio del 2025

INTRODUCCION

En la era digital actual, donde el consumo de contenido multimedia, especialmente la música, ha alcanzado niveles sin precedentes a través de plataformas de streaming, la capacidad de ofrecer experiencias personalizadas se ha vuelto fundamental. La vasta cantidad de opciones disponibles para los usuarios, si bien es una ventaja, también presenta el desafío de la sobrecarga de información, haciendo que la búsqueda de nueva música que se adapte a los gustos individuales sea una tarea ardua. Es en este contexto donde los sistemas de recomendación emergen como herramientas indispensables para filtrar, organizar y sugerir contenido relevante, mejorando significativamente la experiencia del usuario y fomentando el descubrimiento.

El presente informe detalla el diseño, implementación y resultados de un Sistema Básico de Recomendación de Música, concebido para abordar este desafío de personalización. Nuestra propuesta se centra en la aplicación de un Análisis OLAP Simplificado (Online Analytical Processing) para extraer patrones y tendencias de consumo musical, lo cual permite generar recomendaciones coherentes y adaptadas. Para gestionar la naturaleza dinámica y a menudo semi-estructurada de los datos musicales y de usuario, se ha optado por una base de datos NoSQL, que ofrece la flexibilidad y escalabilidad necesarias para un entorno de datos en constante crecimiento. A lo largo de este documento, se explorarán los fundamentos teóricos que sustentan el sistema, la arquitectura adoptada, las decisiones técnicas clave, y los resultados obtenidos, proporcionando una visión integral del proyecto y sus implicaciones en la mejora de la interacción del usuario con el contenido musical.

OBJETIVO GENERAL

El objetivo principal es construir una solución de recomendación de música. Para ello, aprovecharemos la capacidad de una base de datos NoSQL como Cassandra, e implementaremos un análisis de tipo OLAP, aunque en su versión más básica.

OBJETIVOS ESPECIFICOS

- **Modelado de Datos NoSQL Simplificado**

Diseñar un esquema NoSQL para almacenar información básica sobre usuarios, canciones y escuchas. Elegir Cassandra como base de datos NoSQL.

- **Implementación de Recomendación Básica**

Implementar un algoritmo simple basado en "Canciones más escuchadas por género" o "Canciones más escuchadas en la misma ciudad del usuario".

- **Análisis OLAP Muy Simplificado**

Identificar dos dimensiones clave: usuario y tiempo. Definir una jerarquía simple para la dimensión tiempo: día → mes. Definir medidas como número de escuchas. Diseñar un cubo OLAP conceptual para análisis. Implementación de Consultas OLAP Básicas, Implementar consultas como "Número total de escuchas por género y mes". Mostrar resultados mediante visualizaciones simples (tablas o gráficos).

DATASET

La fuente de datos inicial para este proyecto está constituida por tres ficheros CSV facilitados por el profesor, los cuales proporcionan información estructurada acerca de las canciones, los usuarios y los registros de escucha. Con el propósito de soportar análisis más sólidos y representativos, se aconseja extender el volumen de cada archivo a 100 registros.

LISTA DE TABLAS

- // Tabla de usuarios por ID (búsqueda directa por UUID)

  `CREATE TABLE IF NOT EXISTS usuarios_id (

  usuario_id uuid PRIMARY KEY,

  nombre text,

  ciudad text

  );`,

- // Tabla de usuarios por ciudad (permite buscar usuarios por ciudad)

  `CREATE TABLE IF NOT EXISTS usuarios_ciudad (

  usuario_id uuid,

  nombre text,

  ciudad text,

  PRIMARY KEY ((ciudad), usuario_id)

  );`,

- // Tabla de canciones por ID (búsqueda directa por UUID)

  `CREATE TABLE IF NOT EXISTS canciones_id (

  cancion_id uuid PRIMARY KEY,

  artista text,

  titulo text,

  genero text,

  duracion decimal

  );`,

- // Tabla de canciones por artista (permite buscar canciones por artista y título)

  `CREATE TABLE IF NOT EXISTS canciones_artista (

  cancion_id uuid,

  artista text,

  titulo text,

  genero text,

  duracion decimal,

  PRIMARY KEY ((artista), titulo, cancion_id)

  );`,

- // Tabla de escuchas diarias por usuario (registra cada evento de escucha individual)

  `CREATE TABLE IF NOT EXISTS escuchas_diarias_usuario (

  usuario_id uuid,

  cancion_id uuid,

  fecha_escucha date,

  minutos_escucha decimal,

  cancion_titulo text,

  cancion_artista text,

  cancion_genero text,

  PRIMARY KEY ((usuario_id), fecha_escucha, cancion_id)

) WITH CLUSTERING ORDER BY (fecha_escucha DESC);`,

- // Tabla de totales acumulados de minutos escuchados por usuario y canción

  `CREATE TABLE IF NOT EXISTS minutos_usuario (

  usuario_id uuid,

  cancion_id uuid,

  total_minutos decimal,

PRIMARY KEY ((usuario_id), cancion_id)

);`,

- // Tabla de top canciones más escuchadas por usuario

  `CREATE TABLE IF NOT EXISTS mas_escuchados_usuario (

  usuario_id uuid,

  cancion_id uuid,

  total_minutos decimal,

  PRIMARY KEY ((usuario_id), total_minutos, cancion_id)

) WITH CLUSTERING ORDER BY (total_minutos DESC);`,

- // Tabla de escuchas diarias por género

  `CREATE TABLE IF NOT EXISTS escuchas_diarias_genero (

  cancion_id uuid,

  fecha_escucha date,

  genero_cancion text,

  artista_cancion text,

  titulo_cancion text,

  minutos_escucha decimal,

  PRIMARY KEY ((genero_cancion), fecha_escucha, cancion_id)

) WITH CLUSTERING ORDER BY (fecha_escucha DESC);`,

- // Tabla de totales acumulados de minutos escuchados por género y canción

  `CREATE TABLE IF NOT EXISTS minutos_genero (

  genero_cancion text,

  cancion_id uuid,

  total_minutos decimal,

PRIMARY KEY ((genero_cancion), cancion_id)

);`,

- // Tabla de top canciones más escuchadas por género

  `CREATE TABLE IF NOT EXISTS mas_escuchados_genero (

  genero_cancion text,

  cancion_id uuid,

  total_minutos decimal,

  PRIMARY KEY ((genero_cancion), total_minutos, cancion_id)

) WITH CLUSTERING ORDER BY (total_minutos DESC);`,

- // Tabla de escuchas diarias por ciudad

  `CREATE TABLE IF NOT EXISTS escuchas_diarias_ciudad (

  cancion_id uuid,

  ciudad text,

  fecha_escucha date,

  genero_cancion text,

  artista_cancion text,

  titulo_cancion text,

  minutos_escucha decimal,

  PRIMARY KEY ((ciudad), fecha_escucha, cancion_id)

) WITH CLUSTERING ORDER BY (fecha_escucha DESC);`,

- // Tabla de totales acumulados de minutos escuchados por ciudad y canción

  `CREATE TABLE IF NOT EXISTS minutos_ciudad (

  ciudad text,

  cancion_id uuid,

total_minutos decimal,

PRIMARY KEY ((ciudad), cancion_id)

);`,

- // Tabla de top canciones más escuchadas por ciudad

  `CREATE TABLE IF NOT EXISTS mas_escuchados_ciudad (

  ciudad text,

  cancion_id uuid,

  total_minutos decimal,

  PRIMARY KEY ((ciudad), total_minutos, cancion_id)

) WITH CLUSTERING ORDER BY (total_minutos DESC);`,

REQUISITOS OLAP

1. Hechos (Fact Table)

El hecho principal es la "escucha de una canción", que se almacena en la tabla **escuchas_diarias_usuario**. Cada registro representa cuántos minutos un usuario escuchó una canción en una fecha específica.

- Tabla de hechos:

- escuchas_diarias_usuario

  Claves: usuario_id, fecha_escucha, cancion_id

  Medida: minutos_escucha

Atributos desnormalizados: cancion_titulo, cancion_artista, cancion_genero

2. Dimensiones

Las dimensiones principales del cubo son:

- Usuario (usuario_id, nombre, ciudad)
- Canción (cancion_id, titulo, artista, genero, duracion)
- Fecha (fecha_escucha)
- Ciudad (a través de la dimensión usuario)
- Género (a través de la dimensión canción)

3. Medidas

- La medida principal es:

Minutos escuchados (minutos_escucha)

4. Tablas de Agregados (Materialized Views)

Para acelerar consultas OLAP típicas, se crean tablas agregadas (precalculadas) para distintos cortes del cubo:

Por usuario y canción:

minutos_usuario y mas_escuchados_usuario

Por género y canción:

minutos_genero y mas_escuchados_genero

Por ciudad y canción:

minutos_ciudad y mas_escuchados_ciudad

Por género y fecha:

escuchas_diarias_genero

Por ciudad y fecha:

escuchas_diarias_ciudad

ESTRUCTURA DEL CUBO OLAP

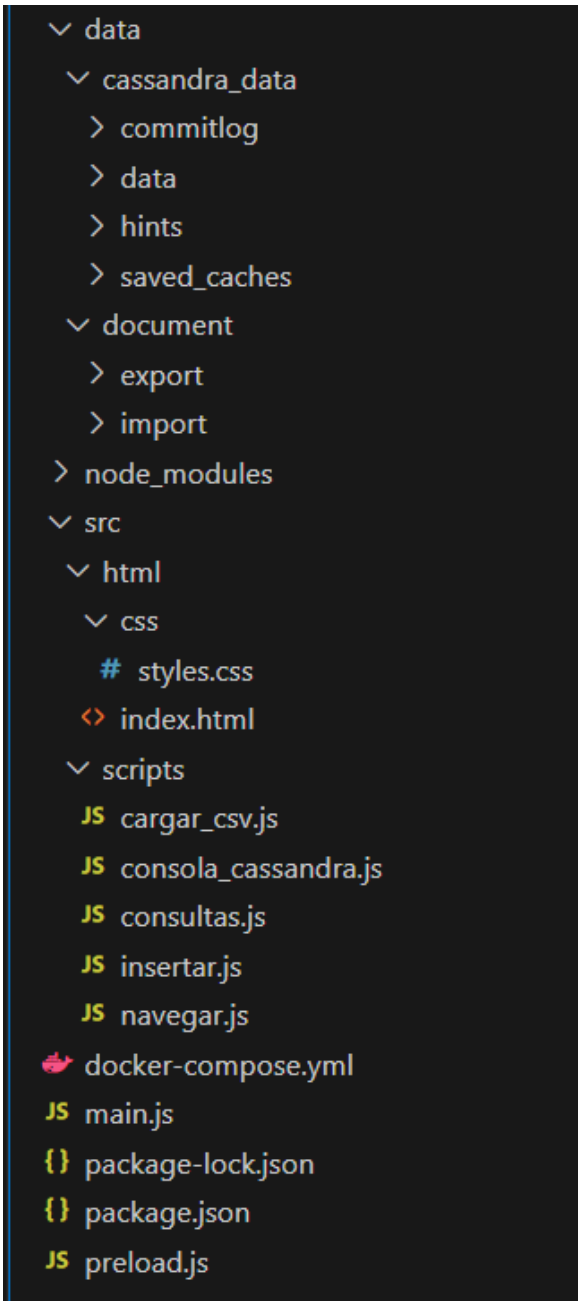
El cubo OLAP puede representarse así:

- Hecho: Minutos escuchados
- Dimensiones: Usuario, Canción, Fecha, Ciudad, Género
- Medidas: Suma de minutos escuchados

Ejemplo de consultas OLAP posibles:

- Total de minutos escuchados por usuario, por género, por ciudad, por fecha, por canción.
- Top N canciones más escuchadas por usuario, género o ciudad.
- Evolución diaria de escuchas por género o ciudad.

DISEÑO GENERAL DEL SISTEMA



1. Fuente de Datos

- Colección de música: Información de canciones, géneros, artistas, popularidad.

- Preferencias de usuarios: Historial de reproducción, calificaciones, interacciones.

- Datos externos (opcional): Tendencias globales, redes sociales, integración con APIs de música.

2. Base de Datos NoSQL

- Cassandra (columnar) para almacenamiento flexible.

- Tablas o colecciones clave:

- usuarios: ID, historial de reproducción, géneros favoritos.

- canciones: ID, artista, duración, popularidad.

- interacciones: registros de "me gusta", comentarios y frecuencia de reproducción.

- recomendaciones: sugerencias generadas por el sistema.

3. Módulo de Recomendación

- Basado en filtrado colaborativo: Encuentra patrones entre usuarios con gustos similares.

- Análisis de contenido: Relaciona características de canciones con preferencias individuales.

- Híbrido: Combina los dos enfoques para mayor precisión.

4. Procesamiento OLAP

- Cubos de datos en Apache Druid o herramientas similares.

- Permite consultas rápidas sobre tendencias de reproducción, canciones más escuchadas por categoría, relación entre géneros y preferencias de usuarios.

- Métricas clave:

- Canciones más populares en un periodo.

- Análisis de géneros por región o segmento de usuarios.

- Frecuencia de reproducción por hora del día.

5. Interfaz de Usuario

- HTML, CSS

CASOS DE USOS

El diseño e implementación de este sistema se basaron en tres funcionalidades principales, las cuales detallamos a continuación:

1. Registro de reproducciones musicales:

- Quién lo usa: Un usuario.
- Cómo funciona: Cuando un usuario reproduce una canción, el sistema graba automáticamente este evento, guardando la fecha, el usuario que la escuchó y la canción en cuestión.

2. Recomendación de canciones por ubicación (ciudad):

- Quién lo usa: Un usuario.
- Cómo funciona: El sistema le presenta al usuario un listado con las canciones más populares o escuchadas en su ciudad actual.

3. Consulta de escuchas por categoría (género) y periodo (mes):

- Quién lo usa: Un administrador o analista.
- Cómo funciona: Para fines de análisis de datos, el sistema genera y entrega el total de reproducciones, organizadas por género musical y por mes.

JUSTIFICACION TECNICA

Cassandra

Se optó por Apache Cassandra para ser la base de datos central de nuestro proyecto. Su elección se justificó por su habilidad superior para gestionar cantidades masivas de datos de forma distribuida, lo que le permite escalar horizontalmente de manera ilimitada. En el contexto de un sistema de recomendación musical, necesitábamos una infraestructura que pudiera soportar un flujo constante de inserciones y recuperaciones de datos, derivado del registro incesante de reproducciones y de las consultas analíticas. Adicionalmente, las capacidades de replicación automática y la alta disponibilidad continua de Cassandra resultan ser cualidades irremplazables para una aplicación moderna.

Docker Compose

Para la configuración de nuestro entorno de desarrollo, nos apoyamos en Docker Compose. Esta herramienta facilitó enormemente la puesta en marcha y conexión de los contenedores esenciales, como el servidor de Cassandra y el 'backend' de Node.js. Esto nos aseguró una configuración consistente y portátil, lo que simplificó significativamente el desarrollo y las futuras implementaciones. Además, garantizó la igualdad entre los entornos de desarrollo local y de producción, minimizando así los posibles problemas de configuración.

Node.js

El 'backend' se desarrolló utilizando Node.js, una elección justificada por su óptimo rendimiento en operaciones de E/S y su facilidad de integración con Cassandra mediante librerías como cassandra-driver. Esta fusión tecnológica confirió al proyecto un diseño modular, eficiente y sumamente escalable. Node.js es solo como una extensión pa conectar JS con el electron

IMPLEMENTACION DE ALGORITMO

El algoritmo utilizado en este proyecto se caracteriza por su simplicidad intencionada, centrada en la ubicación del usuario. Su funcionamiento se puede resumir de la siguiente manera:

• Se determina la ciudad registrada del usuario.

• Con base en esta ciudad, se consulta una base de datos para encontrar las canciones más populares en esa área.

• El sistema genera una lista de canciones locales recomendadas.

Este método permite personalizar la experiencia del usuario sin necesidad de realizar cálculos complicados o manejar grandes cantidades de datos interconectados. La implementación se lleva a cabo en el backend con Node.js y consultas en CQL (Cassandra Query Language) para agrupar y contabilizar reproducciones por ciudad.

Además, este modelo de recomendaciones puede ampliarse en el futuro para incluir otros criterios, como el género musical preferido, artistas favoritos o interacciones previas del usuario

CODIGO FUENTE

Configuración de cassandra

```
const { app, BrowserWindow, ipcMain } = require('electron')

const fs = require('fs');
const path = require('path');
// npm install cassandra-driver
const cassandra = require('cassandra-driver');
// npm install readline-sync
const readlineSync = require('readline-sync');
// npm install csv-parse
const { parse: csvParse } = require('csv-parse/sync');

const Uuid = cassandra.types.Uuid; // Para manejar valores UUID en Cassandra
const BigDecimal = cassandra.types.BigDecimal; // Para manejar valores decimales grandes
const LocalDate = cassandra.types.LocalDate; // Para manejar fechas sin hora (YYYY-MM-DD)

const prefix = `  Console  |`; /// Decorador
const ip = 'localhost'; // IP del Cassandra

// Configuración de conexión a Cassandra
const client = new cassandra.Client({
    contactPoints: [`${ip}`],
    localDataCenter: 'datacenter1', // El data center en docker-compose.yml
});
```

Funcion insertar

```
/**
 * Bucle del menú de inserciones. Permite insertar usuarios, canciones y escuchas.
 */
async function menuInserciones() {
    let volver = false;
    while (!volver) {
        const opcion = mostrarMenuInserciones();
        switch (opcion) {
            case '1': {
                const nombre = readlineSync.question('Nombre del usuario: ');
                const ciudad = readlineSync.question('Ciudad: ');
                await insertarUsuario(nombre, ciudad);
                break;
            }
            case '2': {
                const artista = readlineSync.question('Artista: ');
                const titulo = readlineSync.question('Título: ');
                const genero = readlineSync.question('Género: ');
                const duracion = readlineSync.question('Duración (minutos): ');
                await insertarCancion({ artista, titulo, genero, duracion });
                break;
            }
            case '3': {
                // Insertar escucha
                const usuario_id = readlineSync.question('UUID del usuario: ');
                const cancion_id = readlineSync.question('UUID de la canción: ');
                const fecha_escucha = readlineSync.question('Fecha de escucha (YYYY-MM-DD, opcional, enter para hoy): ');
                // Buscar detalles usuario y canción
                const usuario = await buscarUsuarioPorId({ usuario_id });
                const cancion = await buscarCancionPorId({ cancion_id });
                if (!usuario) {
                    console.log('Usuario no encontrado.');
                    break;
                }
```

```
/**
 * Inserta una nueva canción en las tablas 'canciones_id' y 'canciones_artista' usando un batch.
 * @param {object} songData Objeto con { artista, titulo, genero, duracion }
 * @returns {Promise<Uuid>}
 */
async function insertarCancion(songData) {
    const cancionId = Uuid.random();
    const duracion = songData.duracion;
    const duracionDecimal = BigDecimal.fromString(duracion.toString());

    const queries = [
        {
            query: `INSERT INTO canciones_id (cancion_id, artista, titulo, genero, duracion) VALUES (?, ?, ?, ?, ?)`,
            params: [cancionId, songData.artista, songData.titulo, songData.genero, duracionDecimal]
        },
        {
            query: `INSERT INTO canciones_artista (cancion_id, artista, titulo, genero, duracion) VALUES (?, ?, ?, ?, ?)`,
            params: [cancionId, songData.artista, songData.titulo, songData.genero, duracionDecimal]
        }
    ];

    try {
        await client.batch(queries, { prepare: true, logged: false });
        console.log(`Canción '${songData.titulo}' de '${songData.artista}' (${cancionId}) insertada en ambas tablas.`);
        return cancionId;
    } catch (err) {
        console.error(`Error al insertar canción '${songData.titulo}':`, err);
        throw err;
    }
}
```

```
542  async function registrarEscucha(usuarioId, cancionId, minutosEscucha, songDetails, ciudadUsuario, fecha) {
543      let fechaActual;
544
545      if (fecha) {
546          if (typeof fecha === 'string') {
547              fechaActual = LocalDate.fromString(fecha);
548          } else {
549              fechaActual = fecha;
550          }
551      } else {
552          fechaActual = LocalDate.fromDate(new Date());
553      }
554      const queries = [];
555
556      // --- Lógica para 'escuchas_diarias_usuario'  ---
557      let oldMinutosEscuchaUsuario = 0;
558      try {
559          // 1. **SELECT**: Buscar si ya existe una escucha para este usuario, fecha y canción.
560          const res = await client.execute(
561          `SELECT minutos_escucha FROM escuchas_diarias_usuario WHERE usuario_id = ? AND fecha_escucha = ? AND cancion_id = ?`,
562          [usuarioId, fechaActual, cancionId],
563          { prepare: true }
564          );
565          if (res.rows.length > 0) {
566          oldMinutosEscuchaUsuario = res.rows[0].minutos_escucha.toNumber(); // Convertir BigDecimal a número
567          }
568      } catch (err) {
569          console.warn(`Advertencia: No se pudo obtener minutos_escucha de escuchas_diarias_usuario. Error: ${err.message}`);
570      }
```

Funciones de búsqueda

```
async function buscarCancionesPorArtista(artista) {
    try {
        const res = await client.execute(
            'SELECT * FROM canciones_artista WHERE artista = ?',
            [artista],
            { prepare: true }
        );
        const canciones = res.rows.map(c => ({
            CancionID: c.cancion_id.toString(),
            Artista: c.artista,
            Titulo: c.titulo,
            Genero: c.genero,
            Duracion: c.duracion.toString()
        }));

        return canciones;
    } catch (err) {
        console.error(`Error al buscar canciones por artista '${artista}':`, err);
        return [];
    }
}
```

Búsqueda por ciudad:

```
async function buscarUsuariosPorCiudad(ciudad) {
    try {
        const res = await client.execute(
            'SELECT * FROM usuarios_ciudad WHERE ciudad = ?',
            [ciudad],
            { prepare: true }
        );
        const usuarios = res.rows.map(u => ({
            UsuarioID: u.usuario_id.toString(),
            Nombre: u.nombre,
            Ciudad: u.ciudad
        }));

        return usuarios;
    } catch (err) {
        console.error(`Error al buscar usuarios por ciudad '${ciudad}':`, err);
        return [];
    }
}
```

Funciones para eliminar registro :

```
async function eliminarUsuario(usuarios) {
    if (!usuarios || usuarios.length === 0) {
        console.log('No hay usuarios para eliminar.');
        return;
    }
    const index = readlineSync.questionInt('Ingrese el índice del usuario a eliminar: ');
    const usuario = usuarios.find(u => u.idx === index);
    if (!usuario) {
        console.log('Índice inválido.');
        return;
    }
    const usuarioId = cassandra.types.Uuid.fromString(usuario.ID);
    const queries = [
        {
            query: 'DELETE FROM usuarios_id WHERE usuario_id = ?',
            params: [usuarioId]
        },
        {
            query: 'DELETE FROM usuarios_ciudad WHERE ciudad = ? AND usuario_id = ?',
            params: [usuario.Ciudad, usuarioId]
        }
    ];
    await client.batch(queries, { prepare: true, logged: false });
    console.log('Usuario eliminado de ambas tablas:');
    console.table([usuario]);
}
```

```
async function eliminarCancion(canciones) {
    if (!canciones || canciones.length === 0) {
        console.log('No hay canciones para eliminar.');
        return;
    }
    const index = readlineSync.questionInt('Ingrese el índice de la canción a eliminar: ');
    const cancion = canciones.find(c => c.idx === index);
    if (!cancion) {
        console.log('Índice inválido.');
        return;
    }
    const cancionId = cassandra.types.Uuid.fromString(cancion.ID);
    const queries = [
        {
            query: 'DELETE FROM canciones_id WHERE cancion_id = ?',
            params: [cancionId]
        },
        {
            query: 'DELETE FROM canciones_artista WHERE artista = ? AND titulo = ? AND cancion_id = ?',
            params: [cancion.Artista, cancion.Titulo, cancionId]
        }
    ];
    await client.batch(queries, { prepare: true, logged: false });
    console.log('Canción eliminada de ambas tablas:');
    console.table([cancion]);
}
```

Funciones de csv

```
 * @param {string} filePath Ruta al archivo CSV.
 */
async function insertarUsuariosDesdeCSV(filePath) {
    try {
        const absPath = path.resolve(filePath);
        const csvContent = fs.readFileSync(absPath, 'utf8');
        const records = csvParse(csvContent, {
            columns: true,
            skip_empty_lines: true,
            trim: true

        });

        if (!Array.isArray(records) || records.length === 0) {
            console.log('El archivo CSV no contiene registros.');
            return;
        }

        const queries = [];
        for (const row of records) {
            if (!row.usuario_id || !row.nombre || !row.ciudad) continue;
            let usuarioId;
            try {
                usuarioId = Uuid.fromString(row.usuario_id);
            } catch {
                console.warn(`UUID inválido para usuario: ${row.usuario_id}, se omite.`);
                continue;
            }
            queries.push({
                query: 'INSERT INTO usuarios_id (usuario_id, nombre, ciudad) VALUES (?, ?, ?)',
                params: [usuarioId, row.nombre, row.ciudad]
            });
            queries.push({
                query: 'INSERT INTO usuarios_ciudad (usuario_id, nombre, ciudad) VALUES (?, ?, ?)',
                params: [usuarioId, row.nombre, row.ciudad]
            });
        }
```

```
                queries.push({
                    query: 'INSERT INTO usuarios_id (usuario_id, nombre, ciudad) VALUES (?, ?, ?)',
                    params: [usuarioId, row.nombre, row.ciudad]
                });
                queries.push({
                    query: 'INSERT INTO usuarios_ciudad (usuario_id, nombre, ciudad) VALUES (?, ?, ?)',
                    params: [usuarioId, row.nombre, row.ciudad]
                });
            }

            if (queries.length === 0) {
                console.log('No se encontraron usuarios válidos en el CSV.');
                return;
            }

            await client.batch(queries, { prepare: true, logged: false });
            return`Se insertaron ${queries.length / 2} usuarios desde el archivo CSV.`;
        } catch (err) {
            console.error('Error al insertar usuarios desde CSV:', err);
        }
    }
```

Inserta escuchas de canciones de usuarios desde un archivo CSV

```
 * @param {string} filePath Ruta al archivo CSV.
 */
async function insertarEscuchasDesdeCSV(filePath) {
    let insertados = 0;
    try {
        const absPath = path.resolve(filePath);
        const csvContent = fs.readFileSync(absPath, 'utf8');
        const records = csvParse(csvContent, {
            columns: true,
            skip_empty_lines: true,
            trim: true
        });

        if (!Array.isArray(records) || records.length === 0) {
            console.log('El archivo CSV no contiene registros.');
            return;
        }


        for (const row of records) {
            const usuario_id = row.usuario_id;
            const cancion_id = row.cancion_id;
            const fecha_escucha = row.fecha_escucha;

            if (!usuario_id || !cancion_id || !fecha_escucha) continue;

            // Buscar usuario y canción
            const usuario = await buscarUsuarioPorId({ usuario_id });
            const cancion = await buscarCancionPorId({ cancion_id });

            if (!usuario || !cancion) {
                console.warn(`No se encontró usuario o canción para usuario_id=${usuario_id}, cancion_id=${cancion_id}.`);
                continue;
            }
```

```
            if (!usuario || !cancion) {
                console.warn(`No se encontró usuario o canción para usuario_id=${usuario_id}, cancion_id=${cancion_id}.`);
                continue;
            }

            // Se asume 1 escucha = duración total de la canción
            const minutosEscucha = cancion.duracion.toNumber ? cancion.duracion.toNumber() : Number(cancion.duracion);

            // Convertir la fecha de escucha a LocalDate usando la constante LocalDate ya definida
            const fechaLocalDate = LocalDate.fromString(fecha_escucha);

            await registrarEscucha(
                usuario_id,
                cancion_id,
                minutosEscucha,
                {
                    titulo: cancion.titulo,
                    artista: cancion.artista,
                    genero: cancion.genero
                },
                usuario.ciudad,
                fechaLocalDate
            );

            insertados++;
        }
        return `Se insertaron ${insertados} escuchas desde el archivo CSV.`;
    } catch (err) {
        console.error('Error al insertar escuchas desde CSV:', err);
    }


}
```

Front index html

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MusicRec</title>
    <link rel="stylesheet" href="./css/styles.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>Sistema de Recomendación Musical - OLAP</h1>
        </header>

        <main>
            <!-- Menú Principal -->
            <section id="main-menu" class="menu-section">
                <h2>Menú Principal</h2>
                <div class="menu-options">
                    <button id="btn-insert">Insertar registros</button>
                    <button id="btn-query">Consultar registros</button>
                    <button id="btn-import">Importar desde CSV</button>
                    <button id="btn-exit">Salir</button>
                </div>
            </section>

            <!-- Insertar Registros -->
            <section id="insert-section" class="menu-section hidden">
                <h2>Insertar Registro</h2>
                <div class="menu-options">
                    <button id="btn-insert-user">Insertar usuario</button>
                    <button id="btn-insert-song">Insertar canción</button>
                    <button id="btn-insert-listen">Insertar escucha</button>
                    <button id="btn-back-insert">Volver</button>
                </div>
```

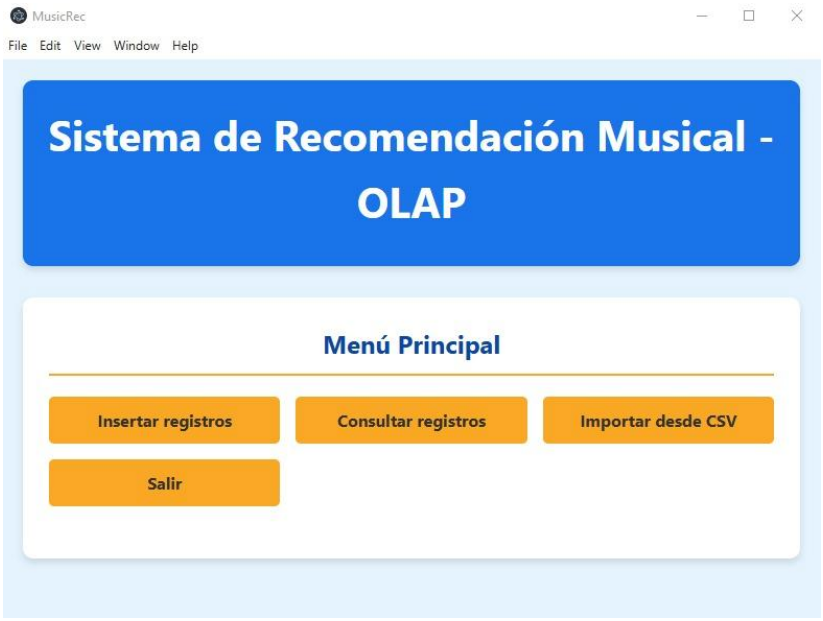
```
        <!-- Insertar Canción -->
        <form id="form-song" class="hidden">
            <h3>Nueva Canción</h3>
            <div class="form-group">
                <label for="song-artist">Artista:</label>
                <input type="text" id="song-artist" required>
            </div>
            <div class="form-group">
                <label for="song-title">Título:</label>
                <input type="text" id="song-title" required>
            </div>
            <div class="form-group">
                <label for="song-genre">Género:</label>
                <input type="text" id="song-genre" required>
            </div>
            <div class="form-group">
                <label for="song-duration">Duración (min):</label>
                <input type="number" step="0.1" id="song-duration" required>
            </div>
            <button type="submit">Guardar</button>
            <p id="msg-song"></p>
        </form>

        <!-- Insertar Escucha -->
        <form id="form-listen" class="hidden">
            <h3>Nueva Escucha</h3>
            <div class="form-group">
                <label for="listen-user">ID Usuario:</label>
                <input type="text" id="listen-user" required>
            </div>
            <div class="form-group">
```

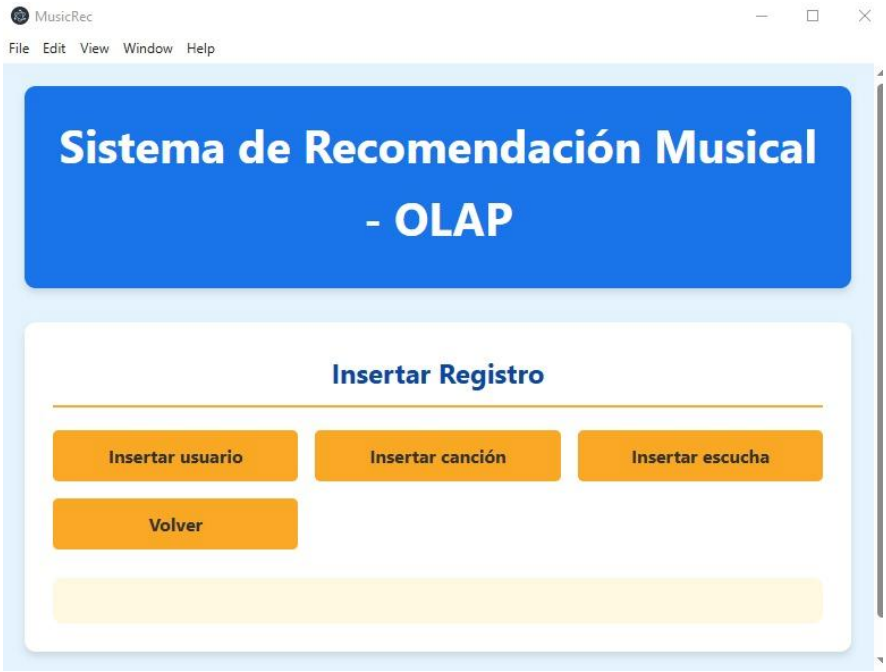
```
98          <!-- Consultar Registros -->
99          <section id="query-section" class="menu-section hidden">
100             <h2>Consultar Registros</h2>
101             <div class="menu-options">
102                 <button id="btn-query-users">Usuarios</button>
103                 <button id="btn-query-songs">Canciones</button>
104                 <button id="btn-query-listens">Escuchas diarias</button>
105                 <button id="btn-query-top">Top Canciones</button>
106                 <button id="btn-back-query">Volver</button>
107         </section>
108
109         <!-- Sección para Top Canciones -->
110         <section id="top-section" class="menu-section hidden">
111             <h2>Top Canciones</h2>
112             <div class="menu-options">
113                 <button id="btn-top-user">Usuario</button>
114                 <button id="btn-top-genre">Género</button>
115                 <button id="btn-top-city">Ciudad</button>
116                 <button id="btn-back-top">Volver</button> <!-- añadido -->
117             </div>
118         </section>
119
120         <!-- Sección para Escuchas Diarias -->
121         <section id="daily-section" class="menu-section hidden">
122             <h2>Escuchas Diarias</h2>
123             <div class="menu-options">
124                 <button id="btn-daily-user">Usuario</button>
```

INTERFAZ

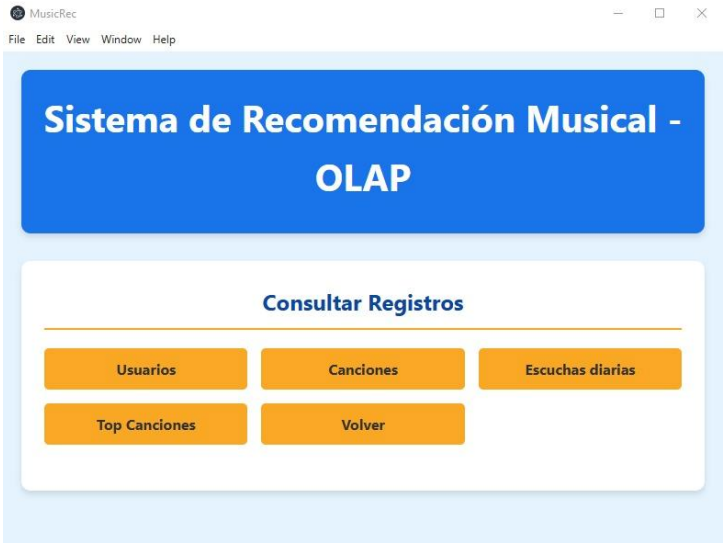
- Interfaz 1 – menú



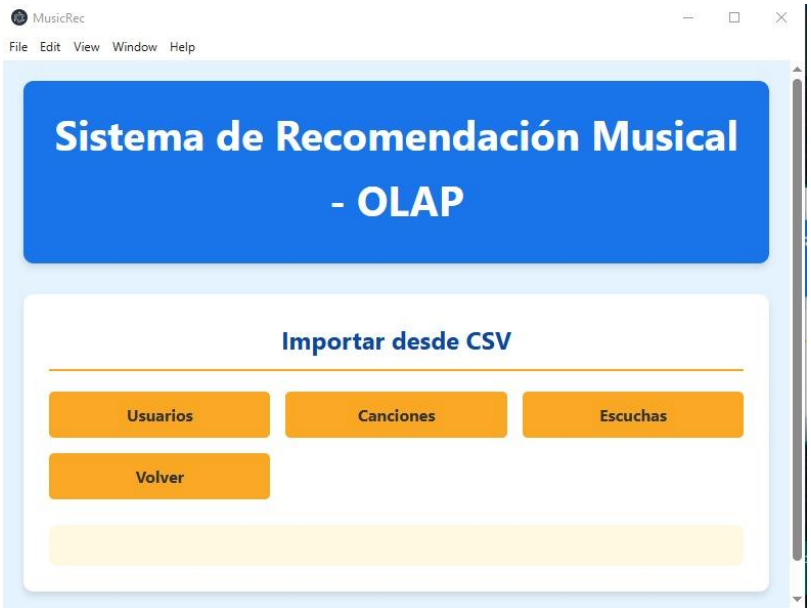
- Insertar registro



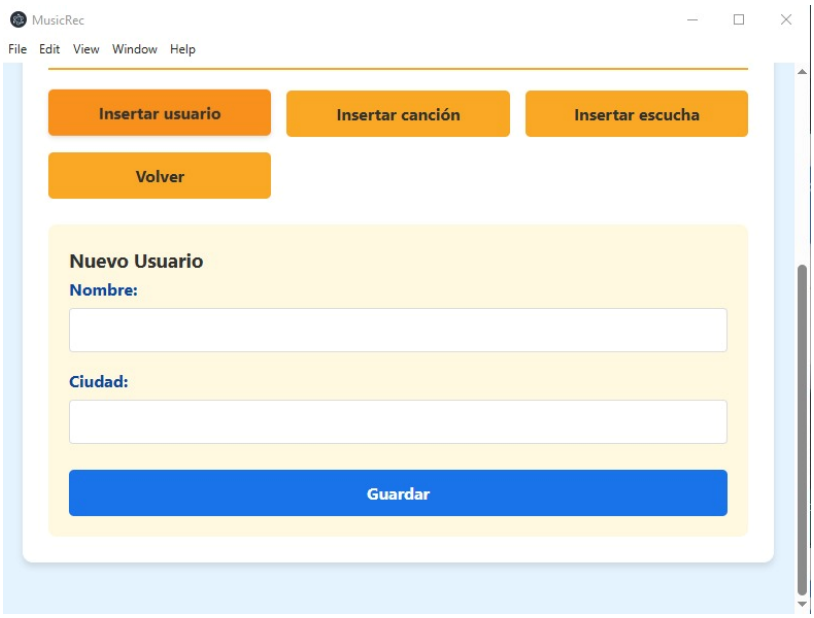
- Consultar registro



- Importar desde CSV

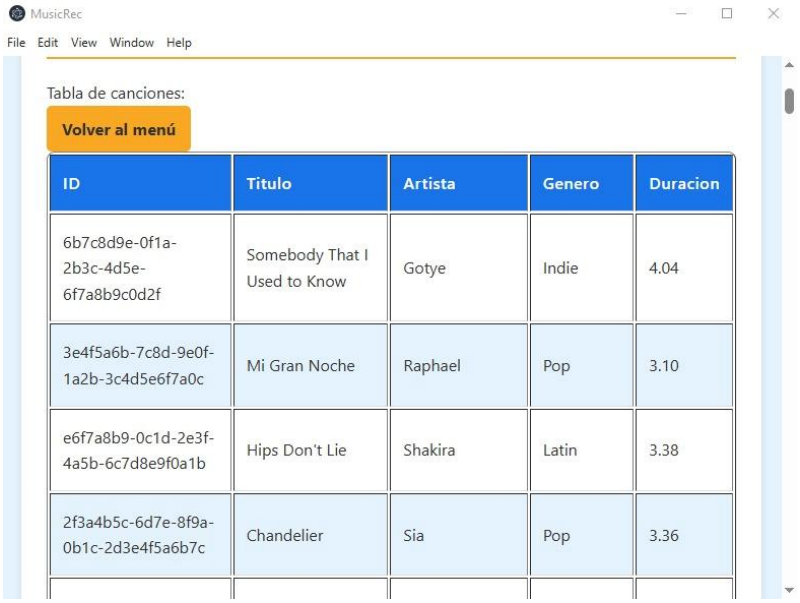


- Interfaz para insertar



PRUEBAS

- Consultas



ID	Titulo	Artista	Genero	Duracion
6b7c8d9e-0f1a-2b3c-4d5e-6f7a8b9c0d2f	Somebody That I Used to Know	Gotye	Indie	4.04
3e4f5a6b-7c8d-9e0f-1a2b-3c4d5e6f7a0c	Mi Gran Noche	Raphael	Pop	3.10
e6f7a8b9-0c1d-2e3f-4a5b-6c7d8e9f0a1b	Hips Don't Lie	Shakira	Latin	3.38
2f3a4b5c-6d7e-8f9a-0b1c-2d3e4f5a6b7c	Chandelier	Sia	Pop	3.36

- consulta de la tabla de usuarios desde el docker

```
cqlsh:recom_musica> SELECT * FROM usuarios_id
                ... ;

 usuario_id                           | ciudad                   | nombre
--------------------------------------+--------------------------+--------------------
 fc3e4567-e89b-12d3-a456-426614174011 |                A Coruña |       Alberto Ruiz
 823e4567-e89b-12d3-a456-426614174007 |                  Murcia |     David Fernandez
 cf3e4567-e89b-12d3-a456-426614174025 |                  Huelva |       Noelia Aguado
 a03e4567-e89b-12d3-a456-426614174056 |              San Roque |       Lucia Navarro
 923e4567-e89b-12d3-a456-426614174008 |                   Palma |       Elena Romero
 fc3e4567-e89b-12d3-a456-426614174052 | Sanlúcar de Barrameda |        Ines Lozano
 fb3e4567-e89b-12d3-a456-426614174010 |                 Vitoria |     Beatriz Suarez
 f93e4567-e89b-12d3-a456-42661417404f |              Puertollano |    Guillermo Pardo
 423e4567-e89b-12d3-a456-426614174003 |                 Sevilla |       Laura Martin
 f73e4567-e89b-12d3-a456-42661417404d |                   Irún |       Samuel Rubio
 ae3e4567-e89b-12d3-a456-426614174018 |                   Jerez |      Andrea Lozano
 ce3e4567-e89b-12d3-a456-426614174024 |              Almendralejo |         Manuel Gil
 ee3e4567-e89b-12d3-a456-426614174044 |                Aranjuez |       Daniela Soto
 a33e4567-e89b-12d3-a456-426614174059 |                Arrecife |         David Ruiz
 aa3e4567-e89b-12d3-a456-426614174014 |                  Oviedo |         Silvia Gil
```

- consulta de la tabla de cancion_id en el Docker

```
(107 rows)
cqlsh:recom_musica> SELECT * FROM canciones_id;

 cancion_id                           | artista           | duracion | genero      | titulo
--------------------------------------+-------------------+----------+-------------+--------------------------------
 6b7c8d9e-0f1a-2b3c-4d5e-6f7a8b9c0d2f |             Gotye |     4.04 |       Indie |    Somebody That I Used to Know
 3e4f5a6b-7c8d-9e0f-1a2b-3c4d5e6f7a0c |           Raphael |     3.10 |         Pop |                  Mi Gran Noche
 e6f7a8b9-0c1d-2e3f-4a5b-6c7d8e9f0a1b |           Shakira |     3.38 |       Latin |                Hips Don't Lie
 2f3a4b5c-6d7e-8f9a-0b1c-2d3e4f5a6b7c |               Sia |     3.36 |         Pop |                     Chandelier
 c7d8e9f0-1a2b-3c4d-5e6f-7a8b9c0d1e5f |      Justin Bieber |     3.18 |         Pop |                        Peaches
 1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c7e |   Enrique Iglesias |     4.03 |       Latin |                       Bailando
 1c2d3e4f-5a6b-7c8d-9e0f-1a2b3c4d5e6f |      The Police |     4.13 |        Rock |           Every Breath You Take
 6f7a8b9c-0d1e-2f3a-4b5c-6d7e8f9a0b1c |         Lady Gaga |     3.57 |         Pop |                     Poker Face
 b9c0d1e2-3f4a-5b6c-7d8e-9f0a1b2c3d4e |   Michael Jackson |     5.57 |         Pop |                       Thriller
 1e2f3a4b-5c6d-7e8f-9a0b-1c2d3e4f5a6b |      Shawn Mendes |     3.11 |         Pop |                       Señorita
 7c8d9e0f-1a2b-3c4d-5e6f-7a8b9c0d1e3a |      Mark Ronson |     4.30 |        Funk |                    Uptown Funk
 5e6f7a8b-9c0d-1e2f-3a4b-5c6d7e8f9a1c |              CNCO |     3.43 |       Latin |                Reggaetón Lento
 4f5a6b7c-8d9e-0f1a-2b3c-4d5e6f7a8b9c |         Ed Sheeran |     3.53 |         Pop |                     Shape of You
 2d3e4f5a-6b7c-8d9e-0f1a-2b3c4d5e6f7a |       The Killers |     3.43 | Alternative |                  Mr. Brightside
 3c4d5e6f-7a8b-9c0d-1e2f-3a4b5c6d7e9a |            Maluma |     3.49 |    Reggaeton |                  Felices los 4
```

CONCLUSION

Hemos explorado las características y beneficios de Apache Cassandra como una solución de base de datos basada en columnas. Cassandra se destaca por su capacidad para manejar grandes volúmenes de datos distribuidos de manera eficiente, ofreciendo alta disponibilidad y escalabilidad horizontal. A lo largo del análisis, hemos observado que su arquitectura descentralizada y su modelo de consistencia configurable permiten a las organizaciones adaptarse a diversas necesidades y cargas de trabajo. Además, la capacidad de Cassandra para realizar escrituras rápidas y su diseño orientado a columnas lo convierten en una opción ideal para aplicaciones que requieren un acceso rápido a datos analíticos y transaccionales. La replicación y la tolerancia a fallos son otras características clave que garantizan la integridad y disponibilidad de los datos, incluso en entornos de alta demanda.

Sin embargo, es importante tener en cuenta que la implementación de Cassandra requiere un entendimiento profundo de su modelo de datos y un diseño cuidadoso para aprovechar al máximo sus capacidades. La curva de aprendizaje puede ser un desafío, pero los beneficios en términos de rendimiento y escalabilidad justifican el esfuerzo