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:

MANSOUR ORTEGA

HECTOR VASQUEZ

LUIS ZAMORA

Puerto Ordaz 06 de junio del 2025

Introducción

Este informe explica cómo se ideó, se armó y se puso en marcha un sistema sencillo para dar sugerencias de música. Se usaron cosas modernas y se pensó en enseñar cómo funciona. La idea junta una base de datos diferente (Apache Cassandra), un motor hecho con Node. js y una página web fácil de usar.

Este plan es como una forma fácil de entender cómo funciona el negocio, usando preguntas juntas que parecen funciones OLAP (Procesamiento Analítico en Línea). Estas se enfocan en cosas importantes como el tipo de música y el tiempo. Para ello, se usó una forma de armar el sistema que es como piezas, de tal forma sería más fácil de cambiar y crecer.

El trabajo se partió en tres momentos claros: pensar cómo sería y cómo funcionaría, hacerlo de verdad y, al final, revisarlo y escribir cómo se hizo. Se hicieron cosas como dibujar cómo serían los datos, preparar el lugar para trabajar con Docker Compose, meter datos desde archivos CSV, construir partes para dar recomendaciones y hacer preguntas OLAP básicas.

Este informe incluye cada una de las partes del sistema, dando una explicación al porqué se eligieron, cómo están organizadas, cómo funcionan en general y cómo se ven los resultados.

Objetivo General

Diseñar e implementar un sistema de recomendación de música con un análisis OLAP simplificado, utilizando una base de datos NoSQL (Cassandra).

Objetivos Específicos

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.
- Justificar la elección basada en la escalabilidad y eficiencia para manejar grandes volúmenes de datos.

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 Utilizado

El sistema se alimenta inicialmente de tres archivos CSV proporcionados por el docente, con datos estructurados en torno a canciones, usuarios y escuchas. Se recomienda ampliar cada archivo a 100 registros para análisis más robustos.

Canciones (canciones.csv)

cancion_id,titulo,artista,genero
1,Bohemian Rhapsody,Queen,Rock
2,Like a Prayer,Madonna,Pop
3,Smells Like Teen Spirit,Nirvana,Grunge
4,Hey Jude,The Beatles,Rock
5,Billie Jean,Michael Jackson,Pop

Usuarios (usuarios.csv)

usuario_id,nombre,ciudad
101,Ana Perez,Caracas
102,Carlos Gomez,Bogota
103,Laura Torres,Mexico DF
104,Javier Ruiz,Caracas
105,Sofia Martinez,Bogota

Escuchas (escuchas.csv)

usuario_id,cancion_id,fecha_escucha
101,1,2024-01-05
102,2,2024-01-05
101,3,2024-01-10
103,1,2024-01-12
104,4,2024-01-15
105,5,2024-01-20
102,1,2024-01-25
103,4,2024-02-01

Estos datos iniciales permiten probar y validar la funcionalidad básica del sistema. La expansión a 100 registros por archivo facilitará evaluaciones más detalladas y significativas en la etapa de pruebas.

Justificación Técnica

Elección de Cassandra como base de datos NoSQL

Cassandra fue seleccionada como el núcleo de datos, gracias a su capacidad sobresaliente para administrar cantidades enormes de información de manera descentralizada y con la posibilidad de crecer sin límites. Dentro del marco del sistema para recomendar música, se requiere una estructura capaz de aguantar una gran cantidad de adiciones y extracciones de datos, algo común debido al registro incesante de reproducciones y las indagaciones analíticas. Además, Cassandra facilita la réplica automática y asegura una disponibilidad continua, cualidades esenciales para una aplicación actual.

Uso de Docker Compose

Para organizar el ambiente de desarrollo, nos apoyamos en Docker Compose. Con él, fue sencillo activar y vincular los contenedores que requeríamos, como el servidor de Cassandra y el 'backend' de Node. js. Esto nos dio la tranquilidad de tener una configuración que se puede replicar y llevar a cualquier parte, simplificando el desarrollo y las implementaciones que vengan. Aparte, nos da la certeza de que los ambientes locales y los de producción serán iguales, disminuyendo los problemas de configuración.

Uso de Node.js

Se optó por Node. js para el backend porque maneja de maravilla las operaciones de entrada/salida, además de que se vincula sin problemas con Cassandra gracias a librerías como cassandra-driver. Gracias a esta unión de tecnologías, el proyecto puede presumir de un diseño que es modular, eficiente y, sobre todo, muy escalable.

Estructura del Proyecto

El proyecto se organiza en una estructura clara que separa los componentes principales del sistema en backend, frontend y configuración de entorno.

```
music-recommendation-project/
├── docker-compose.yml
├── backend/
│   ├── Dockerfile
│   ├── package.json
│   ├── index.js
│   ├── cassandra-setup.js
│   ├── import_data.js
│   └── data/
│       ├── canciones.csv
│       ├── usuarios.csv
│       └── escuchas.csv
├── frontend/
│   ├── index.html
│   ├── app.js
│   └── styles.css
└── README.md
    └── docs/
```

- **docker-compose.yml**: Orquesta los servicios necesarios para la aplicación (Cassandra y el backend de Node.js).
- **backend/**: Contiene toda la lógica del servidor, conexión a Cassandra y carga de datos.
    - index.js: Punto de entrada del servidor backend.
    - cassandra-setup.js: Encargado de crear las tablas necesarias en Cassandra.
    - import_data.js: Automatiza la carga de datos desde los archivos CSV a la base de datos.
    - data/: Contiene los archivos CSV con datos iniciales de usuarios, canciones y escuchas.
- **frontend/**: Interfaz básica de usuario para visualizar consultas OLAP.
    - index.html: Estructura básica del sitio web.
    - app.js: Lógica de cliente que realiza peticiones al backend y renderiza resultados.
    - styles.css: Estilos simples para la visualización.
- **README.md y docs/**: Documentación del proyecto para facilitar instalación y uso.

Esta organización modular permite mantener un desarrollo ordenado y facilita futuras extensiones del sistema.

Casos de Uso

A continuación, se presentan tres casos de uso básicos que orientaron el diseño e implementación del sistema:

1. **Registrar escucha de una canción**
   - Actor: Usuario
   - Descripción: El usuario reproduce una canción. El sistema almacena la escucha junto con la fecha, el usuario y la canción correspondiente.
2. **Recomendar canciones por ciudad**
   - Actor: Usuario
   - Descripción: El sistema ofrece una lista de las canciones más escuchadas en la ciudad del usuario.
3. **Consultar número de escuchas por género y mes**
   - Actor: Administrador o Analista
   - Descripción: El sistema devuelve el total de escuchas agrupadas por género y mes para propósitos analíticos.

Estos casos de uso permitieron mantener el enfoque en funcionalidades concretas, útiles y realistas dentro del alcance del proyecto.

Modelo de Datos NoSQL

El modelo de datos fue diseñado teniendo en cuenta la simplicidad y eficiencia de consulta, estructurando los datos en tres tablas principales:

1. **usuarios**
   - usuario_id (UUID)
   - nombre (TEXT)
   - ciudad (TEXT)
2. **canciones**
   - cancion_id (UUID)
   - titulo (TEXT)
   - artista (TEXT)
   - genero (TEXT)
3. **escuchas**
   - usuario_id (UUID)
   - cancion_id (UUID)
   - fecha_escucha (DATE)

Este modelo permite registrar de manera eficiente las escuchas y realizar consultas relacionadas con la frecuencia de reproducción, géneros preferidos y popularidad de canciones por ubicación o periodo de tiempo. Además, su diseño se alinea con las características de Cassandra, facilitando lecturas rápidas y segmentación de los datos por claves de partición.

Diseño del Cubo OLAP Conceptual

Para el análisis OLAP se diseñó un cubo conceptual basado en las siguientes características:

- **Dimensiones**:
  - **Género**: categoría musical de la canción.
  - **Tiempo**: fecha de la escucha, con jerarquía Día → Mes.
- **Medida Principal**:
  - **Número de escuchas**: total de veces que se escuchó una canción, por género y por periodo de tiempo.

Este cubo permite realizar análisis agregados como:

- Total de escuchas por género y mes.
- Comparación de popularidad de géneros entre diferentes meses.

A pesar de no haber implementado una solución OLAP completa con las debidas herramientas tales como Apache Spark e incluso herramientas de BI, nuestro diseño conceptual permitió la ejecución de consultas básicas sobre la base de datos junto a agrupaciones y conteos emulando funcionalidades OLAP en un entorno simple.

Implementación del Algoritmo de Recomendación

El algoritmo implementado en este proyecto tiene un enfoque deliberadamente simple, basado en la ubicación del usuario. La lógica se resume en lo siguiente:

- Dado un usuario, se identifica su ciudad registrada.
- A partir de esta ciudad, se consulta la base de datos para identificar las canciones más escuchadas en dicha localidad.
- El sistema devuelve una lista de canciones populares locales como recomendación.

Este enfoque permite personalizar la experiencia del usuario sin requerir cálculos complejos o grandes volúmenes de datos interrelacionados. La implementación se realiza en el backend utilizando Node.js y consultas CQL (Cassandra Query Language) para agrupar y contar escuchas por ciudad.

Este modelo de recomendación puede ser fácilmente extendido en el futuro para incluir criterios adicionales como género musical preferido, artistas favoritos o interacciones previas del usuario.

Codigo fuente:

Backend:

Cassandra-setup.js

```
const cassandra = require('cassandra-driver');

const keyspace = 'musicrec';

// Cliente SIN keyspace para crear el keyspace
const client = new cassandra.Client({
  contactPoints: [process.env.CASSANDRA_HOST || 'localhost'],
  localDataCenter: 'datacenter1'
});

async function setupKeyspaceAndTables() {
  try {
    // Crear keyspace si no existe
    const createKeyspaceQuery = `
      CREATE KEYSPACE IF NOT EXISTS ${keyspace}
      WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
    `;
    await client.execute(createKeyspaceQuery);

    // Crear tablas usando el keyspace explícitamente
    const queryUsuarios = `
      CREATE TABLE IF NOT EXISTS ${keyspace}.usuarios (
        usuario_id int PRIMARY KEY,
        nombre text,
        ciudad text
      );
    `;
    await client.execute(queryUsuarios);
```

```
    const queryCanciones = `
      CREATE TABLE IF NOT EXISTS ${keyspace}.canciones (
        cancion_id int PRIMARY KEY,
        titulo text,
        artista text,
        genero text
      );
    `;
    await client.execute(queryCanciones);

    const queryEscuchas = `
      CREATE TABLE IF NOT EXISTS ${keyspace}.escuchas (
        usuario_id int,
        fecha_escucha date,
        cancion_id int,
        PRIMARY KEY (usuario_id, fecha_escucha, cancion_id)
      );
    `;
    await client.execute(queryEscuchas);

    console.log("Keyspace y tablas creadas correctamente.");
  } catch (err) {
    console.error("Error al configurar keyspace y tablas: ", err);
  }
}

// Exporta el cliente y la función
module.exports = { client, setupKeyspaceAndTables, keyspace };
```

import_data.js

```
// Importa los módulos necesarios para manejo de archivos, rutas, CSV y Cassandra
const fs = require('fs'); // Para leer archivos
const path = require('path'); // Para manejar rutas de archivos
const csv = require('csv-parser'); // Para parsear archivos CSV
const { client, setupKeyspaceAndTables } = require('./cassandra-setup'); // Cliente y setup de
Cassandra

// Función genérica para importar un archivo CSV y esperar a que todas las inserciones
terminen
function importCSV(filePath, insertRow) {
  return new Promise((resolve, reject) => {
    const promises = [];
    // Lee el archivo CSV y por cada fila ejecuta insertRow
    fs.createReadStream(filePath)
      .pipe(csv())
      .on('data', (row) => {
        promises.push(insertRow(row)); // Inserta cada fila en la base de datos
      })
      .on('end', async () => {
        try {
          await Promise.all(promises); // Espera a que todas las inserciones terminen
          resolve();
        } catch (err) {
          reject(err);
        }
      })
      .on('error', reject); // Maneja errores de lectura o parseo
```

```
  });
}

// Función principal para importar todos los datos
async function importData() {
  await setupKeyspaceAndTables(); // Asegura que el keyspace y las tablas existen

  // Importar datos de Usuarios
  const usuariosFile = path.join(__dirname, 'data', 'usuarios.csv'); // Ruta al CSV de
usuarios
  await importCSV(usuariosFile, async (row) => {
    const query = 'INSERT INTO musicrec.usuarios (usuario_id, nombre, ciudad) VALUES (?, ?,
?)';
    await client.execute(query, [parseInt(row.usuario_id), row.nombre, row.ciudad], { prepare:
true });
  });
  console.log("Usuarios importados.");

  // Importar datos de Canciones
  const cancionesFile = path.join(__dirname, 'data', 'canciones.csv'); // Ruta al CSV de
canciones
  await importCSV(cancionesFile, async (row) => {
    const query = 'INSERT INTO musicrec.canciones (cancion_id, titulo, artista, genero) VALUES
(?, ?, ?, ?)';
    await client.execute(query, [parseInt(row.cancion_id), row.titulo, row.artista,
row.genero], { prepare: true });
  });
  console.log("Canciones importadas.");

  // Importar datos de Escuchas
  const escuchasFile = path.join(__dirname, 'data', 'escuchas.csv'); // Ruta al CSV de
escuchas
  await importCSV(escuchasFile, async (row) => {
    const query = 'INSERT INTO musicrec.escuchas (usuario_id, fecha_escucha, cancion_id)
VALUES (?, ?, ?)';

    // Convierte la fecha a objeto Date
    const fecha = new Date(row.fecha_escucha);
    await client.execute(query, [parseInt(row.usuario_id), fecha, parseInt(row.cancion_id)], {
prepare: true });
  });
  console.log("Escuchas importadas.");

  // Termina el proceso cuando todo está listo
  process.exit(0);
}

// Ejecuta la importación al correr el script
importData();
```

index.js

```
// Importa los módulos necesarios para el backend
const express = require('express'); // Framework web para Node.js
const bodyParser = require('body-parser'); // Middleware para parsear JSON
const { client, setupKeyspaceAndTables } = require('./cassandra-setup'); // Cliente y setup de
Cassandra
const cors = require('cors'); // Middleware para CORS
const multer = require('multer'); // Middleware para manejo de archivos
const csv = require('csv-parser'); // Parser de archivos CSV
const fs = require('fs'); // Módulo de sistema de archivos
const path = require('path'); // Módulo para rutas de archivos
const upload = multer({ dest: 'uploads/' }); // Configura multer para subir archivos a la
carpeta uploads
const app = express(); // Crea la app de Express
let open;
(async () => {
  open = (await import('open')).default;
})();
/*
  Todas las queries a usuarios, canciones y escuchas ahora usan el prefijo musicrec.
*/
app.use(cors({
  origin: '*'
})); // Permite peticiones desde cualquier origen (CORS)
app.use(bodyParser.json()); // Permite recibir JSON en las peticiones
app.use(bodyParser.urlencoded({ extended: true })); // Permite recibir datos de formularios

const PORT = process.env.PORT || 3000;

// Servir archivos estáticos del frontend
const frontendPath = path.join(__dirname, '../frontend');
app.use(express.static(frontendPath));

// Ruta para servir index.html en la raíz
app.get('/', (req, res) => {
  res.sendFile(path.join(frontendPath, 'index.html'));
});

// Inicializa Cassandra y arranca el servidor
async function startServer() {
  await setupKeyspaceAndTables(); // Crea keyspace y tablas si no existen
  app.listen(PORT, async () => {
    console.log(`Servidor escuchando en http://localhost:${PORT}`);
    // Abrir el navegador automáticamente al iniciar el backend
    if (open) {
      await open(`http://localhost:${PORT}`);
    }
  });
}
startServer();

// --- Endpoint de recomendación personalizada ---
// Devuelve recomendaciones según ciudad o género favorito del usuario
app.get('/recommendation', async (req, res) => {
  try {
    const usuarioId = parseInt(req.query.usuario_id); // ID del usuario
    const modo = req.query.modo || 'ciudad'; // Modo de recomendación
    if (isNaN(usuarioId)) {
```

```
      return res.status(400).json({ error: 'usuario_id debe ser un número' });
    }
    if (modo === 'ciudad') {
      // --- RECOMENDACIÓN POR CIUDAD ---
      // 1. Obtiene la ciudad del usuario
      const userQuery = 'SELECT ciudad FROM musicrec.usuarios WHERE usuario_id = ?';
      const userResult = await client.execute(userQuery, [usuarioId], { prepare: true });
      if (userResult.rowLength === 0) {
        return res.status(404).json({ error: 'Usuario no encontrado' });
      }
      const ciudad = userResult.rows[0].ciudad;
      // 2. Busca todos los usuarios de la misma ciudad
      const allUsersQuery = 'SELECT usuario_id, ciudad FROM musicrec.usuarios';
      const allUsersResult = await client.execute(allUsersQuery);
      const matchingUsers = allUsersResult.rows
        .filter(u => u.ciudad === ciudad)
        .map(u => u.usuario_id);
      // 3. Cuenta las escuchas de los usuarios de esa ciudad
      let songCounts = {};
      for (let uid of matchingUsers) {
        const escuchaQuery = 'SELECT cancion_id FROM musicrec.escuchas WHERE usuario_id = ?';
        const escuchaResult = await client.execute(escuchaQuery, [uid], { prepare: true });
        escuchaResult.rows.forEach(row => {
          songCounts[row.cancion_id] = (songCounts[row.cancion_id] || 0) + 1;
        });
      }
      // 4. Ordena y selecciona las 5 canciones más escuchadas
      const sortedSongs = Object.entries(songCounts)
        .sort((a, b) => b[1] - a[1])
        .slice(0, 5);
      let recommendations = [];
      for (let [cancion_id, count] of sortedSongs) {
        const songQuery = 'SELECT titulo, artista, genero FROM musicrec.canciones WHERE
cancion_id = ?';
        const songResult = await client.execute(songQuery, [parseInt(cancion_id)], { prepare:
true });
        if (songResult.rowLength > 0) {
          recommendations.push({
            cancion_id: cancion_id,
            titulo: songResult.rows[0].titulo,
            artista: songResult.rows[0].artista,
            genero: songResult.rows[0].genero,
            listens: count
          });
        }
      }
      return res.json({ recommendations }); // Devuelve las recomendaciones por ciudad
    } else if (modo === 'genero') {
      // --- RECOMENDACIÓN POR GÉNERO FAVORITO ---
      // 1. Obtiene todas las escuchas del usuario
      const escuchasQuery = 'SELECT cancion_id FROM musicrec.escuchas WHERE usuario_id = ?';
      const escuchasResult = await client.execute(escuchasQuery, [usuarioId], { prepare: true
});
      if (escuchasResult.rowLength === 0) {
        return res.status(404).json({ error: 'El usuario no tiene escuchas registradas' });
      }
      // 2. Cuenta cuántas veces escuchó cada género
      let genreCounts = {};
      for (let row of escuchasResult.rows) {
```

```
        const cancionId = row.cancion_id;
        const songQuery = 'SELECT genero FROM musicrec.canciones WHERE cancion_id = ?';
        const songResult = await client.execute(songQuery, [cancionId], { prepare: true });
        if (songResult.rowLength > 0) {
          const genero = songResult.rows[0].genero;
          genreCounts[genero] = (genreCounts[genero] || 0) + 1;
        }
      }
      // 3. Encuentra el género favorito
      const favoriteGenre = Object.entries(genreCounts).sort((a, b) => b[1] - a[1])[0]?.[0];
      if (!favoriteGenre) {
        return res.status(404).json({ error: 'No se pudo determinar el género favorito' });
      }
      // 4. Busca todas las escuchas de ese género (de todos los usuarios)
      const allEscuchasQuery = 'SELECT cancion_id FROM musicrec.escuchas';
      const allEscuchasResult = await client.execute(allEscuchasQuery);
      let songCounts = {};
      for (let row of allEscuchasResult.rows) {
        const cancionId = row.cancion_id;
        const songQuery = 'SELECT genero FROM musicrec.canciones WHERE cancion_id = ?';
        const songResult = await client.execute(songQuery, [cancionId], { prepare: true });
        if (songResult.rowLength > 0 && songResult.rows[0].genero === favoriteGenre) {
          songCounts[cancionId] = (songCounts[cancionId] || 0) + 1;
        }
      }
      // 5. Ordena y selecciona las 5 canciones más escuchadas de ese género
      const sortedSongs = Object.entries(songCounts)
        .sort((a, b) => b[1] - a[1])
        .slice(0, 5);
      let recommendations = [];
      for (let [cancion_id, count] of sortedSongs) {
        const songQuery = 'SELECT titulo, artista, genero FROM musicrec.canciones WHERE
cancion_id = ?';
        const songResult = await client.execute(songQuery, [parseInt(cancion_id)], { prepare:
true });
        if (songResult.rowLength > 0) {
          recommendations.push({
            cancion_id: cancion_id,
            titulo: songResult.rows[0].titulo,
            artista: songResult.rows[0].artista,
            genero: songResult.rows[0].genero,
            listens: count
          });
        }
      }
      return res.json({ recommendations, genero_favorito: favoriteGenre }); // Devuelve las
recomendaciones por género
    } else {
      return res.status(400).json({ error: 'Modo de recomendación no válido' });
    }
  } catch (err) {
    console.error("Error en /recommendation: ", err);
    res.status(500).json({ error: 'Error interno del servidor' });
  }
});

// --- Endpoint OLAP: escuchas por género y mes ---
// Devuelve un análisis de escuchas agrupadas por género y mes
app.get('/olap/genre-month', async (req, res) => {
```

```
  try {
    const escuchasQuery = 'SELECT usuario_id, cancion_id, fecha_escucha FROM
musicrec.escuchas';
    const escuchasResult = await client.execute(escuchasQuery, [], { prepare: true });
    // Mapea cancion_id a género
    const songsQuery = 'SELECT cancion_id, genero FROM musicrec.canciones';
    const songsResult = await client.execute(songsQuery, [], { prepare: true });
    let songGenreMap = {};
    songsResult.rows.forEach(row => {
      songGenreMap[row.cancion_id] = row.genero;
    });
    let aggregation = {};
    // Agrupa escuchas por género y mes
    escuchasResult.rows.forEach(row => {
      const fecha = new Date(row.fecha_escucha);
      const month = `${fecha.getFullYear()}-${("0" + (fecha.getMonth() + 1)).slice(-2)}`;
      const genero = songGenreMap[row.cancion_id] || 'Desconocido';
      const key = `${genero}_${month}`;
      aggregation[key] = (aggregation[key] || 0) + 1;
    });
    let resultArray = [];
    for (let key in aggregation) {
      const [genero, month] = key.split('_');
      resultArray.push({ genero, month, listens: aggregation[key] });
    }
    res.json({ olap: resultArray }); // Devuelve el análisis OLAP
  } catch (err) {
    console.error("Error en /olap/genre-month: ", err);
    res.status(500).json({ error: 'Error interno del servidor' });
  }
});

// --- Endpoint OLAP: escuchas por género y mes y ciudad ---
// Devuelve un análisis de escuchas agrupadas por género, mes y ciudad
app.get('/olap/genre-month-city', async (req, res) => {
  try {
    // 1. Obtener escuchas con usuario_id, cancion_id, fecha_escucha
    const escuchasQuery = 'SELECT usuario_id, cancion_id, fecha_escucha FROM
musicrec.escuchas';
    const escuchasResult = await client.execute(escuchasQuery, [], { prepare: true });
    // 2. Mapear cancion_id a género
    const songsQuery = 'SELECT cancion_id, genero FROM musicrec.canciones';
    const songsResult = await client.execute(songsQuery, [], { prepare: true });
    let songGenreMap = {};
    songsResult.rows.forEach(row => {
      songGenreMap[row.cancion_id] = row.genero;
    });
    // 3. Mapear usuario_id a ciudad
    const usersQuery = 'SELECT usuario_id, ciudad FROM musicrec.usuarios';
    const usersResult = await client.execute(usersQuery, [], { prepare: true });
    let userCityMap = {};
    usersResult.rows.forEach(row => {
      userCityMap[row.usuario_id] = row.ciudad;
    });
    // 4. Agrupar escuchas por género, mes y ciudad
    let aggregation = {};
    escuchasResult.rows.forEach(row => {
      const fecha = new Date(row.fecha_escucha);
      const month = `${fecha.getFullYear()}-${("0" + (fecha.getMonth() + 1)).slice(-2)}`;
```

```
      const genero = songGenreMap[row.cancion_id] || 'Desconocido';
      const ciudad = userCityMap[row.usuario_id] || 'Desconocido';
      const key = `${genero}_${month}_${ciudad}`;
      aggregation[key] = (aggregation[key] || 0) + 1;
    });
    // 5. Convertir a array de objetos
    let resultArray = [];
    for (let key in aggregation) {
      const [genero, month, ciudad] = key.split('_');
      resultArray.push({ genero, month, ciudad, listens: aggregation[key] });
    }
    res.json({ olap: resultArray });
  } catch (err) {
    console.error("Error en /olap/genre-month-city: ", err);
    res.status(500).json({ error: 'Error interno del servidor' });
  }
});

// --- Endpoint de verificación ---
// Permite comprobar si el backend está corriendo
app.get('/health', (req, res) => {
  res.send('OK');
});

// --- Scripts de utilidad ---
// Permiten iniciar el servidor o importar datos desde package.json
const scripts = {
  start: "node index.js",
  import: "node import_data.js"
};

// --- Endpoint para insertar un usuario ---
// Permite agregar un usuario desde el frontend o por API
app.post('/usuario', async (req, res) => {
  try {
    const { usuario_id, nombre, ciudad } = req.body;
    if (!usuario_id || !nombre || !ciudad) {
      return res.status(400).json({ error: 'Faltan datos para el usuario' });
    }
    const query = 'INSERT INTO musicrec.usuarios (usuario_id, nombre, ciudad) VALUES (?, ?,
?)';
    await client.execute(query, [parseInt(usuario_id), nombre, ciudad], { prepare: true });
    res.json({ mensaje: 'Usuario insertado correctamente' });
  } catch (err) {
    console.error('Error al insertar usuario:', err);
    res.status(500).json({ error: 'Error al insertar usuario' });
  }
});

// --- Endpoint para insertar una canción ---
// Permite agregar una canción desde el frontend o por API
app.post('/cancion', async (req, res) => {
  try {
    const { cancion_id, titulo, artista, genero } = req.body;
    if (!cancion_id || !titulo || !artista || !genero) {
      return res.status(400).json({ error: 'Faltan datos para la canción' });
    }
    const query = 'INSERT INTO musicrec.canciones (cancion_id, titulo, artista, genero) VALUES
(?, ?, ?, ?)';
```

```
    await client.execute(query, [parseInt(cancion_id), titulo, artista, genero], { prepare:
true });
    res.json({ mensaje: 'Canción insertada correctamente' });
  } catch (err) {
    console.error('Error al insertar canción:', err);
    res.status(500).json({ error: 'Error al insertar canción' });
  }
});

// --- Endpoint para insertar una escucha ---
// Permite agregar una escucha desde el frontend o por API
app.post('/escucha', async (req, res) => {
  try {
    const { usuario_id, cancion_id, fecha_escucha } = req.body;
    if (!usuario_id || !cancion_id || !fecha_escucha) {
      return res.status(400).json({ error: 'Faltan datos para la escucha' });
    }
    const fecha = new Date(fecha_escucha);
    const query = 'INSERT INTO musicrec.escuchas (usuario_id, fecha_escucha, cancion_id)
VALUES (?, ?, ?)';
    await client.execute(query, [parseInt(usuario_id), fecha, parseInt(cancion_id)], {
prepare: true });
    res.json({ mensaje: 'Escucha insertada correctamente' });
  } catch (err) {
    console.error('Error al insertar escucha:', err);
    res.status(500).json({ error: 'Error al insertar escucha' });
  }
});

// --- Endpoint para carga masiva de CSV ---
// Permite cargar usuarios, canciones o escuchas desde un archivo CSV
app.post('/cargar-csv/:tipo', upload.single('csv'), async (req, res) => {
  const tipo = req.params.tipo; // Tipo de datos a cargar
  const filePath = req.file.path; // Ruta temporal del archivo
  let insertQuery, insertFn;
  try {
    if (tipo === 'usuarios') {
      insertQuery = 'INSERT INTO musicrec.usuarios (usuario_id, nombre, ciudad) VALUES (?, ?,
?)';
      insertFn = row => client.execute(insertQuery, [parseInt(row.usuario_id), row.nombre,
row.ciudad], { prepare: true });
    } else if (tipo === 'canciones') {
      insertQuery = 'INSERT INTO musicrec.canciones (cancion_id, titulo, artista, genero)
VALUES (?, ?, ?, ?)';
      insertFn = row => client.execute(insertQuery, [parseInt(row.cancion_id), row.titulo,
row.artista, row.genero], { prepare: true });
    } else if (tipo === 'escuchas') {
      insertQuery = 'INSERT INTO musicrec.escuchas (usuario_id, fecha_escucha, cancion_id)
VALUES (?, ?, ?)';

      insertFn = row => client.execute(insertQuery, [parseInt(row.usuario_id), new
Date(row.fecha_escucha), parseInt(row.cancion_id)], { prepare: true });
    } else {
      fs.unlinkSync(filePath);
      return res.status(400).json({ error: 'Tipo de datos no válido' });
    }
    const promises = [];
    fs.createReadStream(filePath)
      .pipe(csv())
```

```
      .on('data', row => promises.push(insertFn(row)))
      .on('end', async () => {
        try {
          await Promise.all(promises);
          fs.unlinkSync(filePath);
          res.json({ mensaje: 'Datos cargados correctamente.' });
        } catch (err) {
          fs.unlinkSync(filePath);
          res.status(500).json({ error: 'Error al insertar datos.' });
        }
      })
      .on('error', err => {
        fs.unlinkSync(filePath);
        res.status(500).json({ error: 'Error al procesar el archivo.' });
      });
  } catch (err) {
    if (fs.existsSync(filePath)) fs.unlinkSync(filePath);
    res.status(500).json({ error: 'Error interno del servidor.' });
  }
});
```

Frontend:

App.js

```
// URL base del backend donde se hacen todas las peticiones
const backendUrl = 'http://localhost:3000';

// Evento para el formulario de recomendaciones personalizadas
// Cuando el usuario envía el formulario de recomendación:
document.getElementById('recommendation-form').addEventListener('submit', async (e) => {
  e.preventDefault(); // Evita que el formulario recargue la página
  const usuarioId = document.getElementById('usuario_id').value; // Obtiene el ID de usuario
  const modo = document.getElementById('modo_recomendacion').value; // Obtiene el modo de
recomendación
  const resultDiv = document.getElementById('recommendation-result'); // Div donde se muestran
los resultados
  resultDiv.innerHTML = 'Cargando recomendaciones...'; // Mensaje de carga

  try {
    // Solicita recomendaciones al backend según usuario y modo
    const response = await
fetch(`${backendUrl}/recommendation?usuario_id=${usuarioId}&modo=${modo}`);
    const data = await response.json(); // Convierte la respuesta a JSON
    if (data.error) {
      // Si hay error, lo muestra
      resultDiv.innerHTML = `<p>Error: ${data.error}</p>`;
    } else {
      // Si hay recomendaciones, las muestra en una lista
      let html = '<h3>Recomendaciones:</h3><ul>';
      data.recommendations.forEach(rec => {
        // Si el modo es ciudad, muestra "Escuchas en tu ciudad"
        if (modo === 'ciudad') {
          html += `<li>${rec.titulo} - ${rec.artista} (Género: ${rec.genero}, Escuchas en tu
ciudad: ${rec.listens})</li>`;
        } else {
          html += `<li>${rec.titulo} - ${rec.artista} (Género: ${rec.genero}, Escuchas:
${rec.listens})</li>`;
```

```
        }
      });
      html += '</ul>';
      resultDiv.innerHTML = html;
    }
  } catch (err) {
    // Si falla la conexión, muestra error
    resultDiv.innerHTML = `<p>Error al conectar al backend.</p>`;
  }
});

// --- Visualización OLAP 2D: Género/Mes/Ciudad ---
document.getElementById('fetch-olap').addEventListener('click', async () => {
  const olapDiv = document.getElementById('olap-result');
  olapDiv.innerHTML = 'Cargando análisis OLAP...';
  try {
    // Solicita el análisis OLAP 2D al backend
    const response = await fetch(`${backendUrl}/olap/genre-month-city`);
    const data = await response.json();
    if (data.error) {
      olapDiv.innerHTML = `<p>Error: ${data.error}</p>`;
    } else {
      // Agrupar por ciudad para mostrar un gráfico por ciudad
      const grouped = {};
      data.olap.forEach(item => {
        if (!grouped[item.ciudad]) grouped[item.ciudad] = [];
        grouped[item.ciudad].push(item);
      });
      // Ordenar por mes (fecha) y luego por género
      Object.keys(grouped).forEach(ciudad => {
        grouped[ciudad].sort((a, b) => {
          // Primero por mes (YYYY-MM)
          if (a.month < b.month) return -1;
          if (a.month > b.month) return 1;
          // Si el mes es igual, por género alfabético
          if (a.genero < b.genero) return -1;
          if (a.genero > b.genero) return 1;
          return 0;
        });
      });
      // Crear un gráfico simple por ciudad usando canvas
      let html = '';
      Object.keys(grouped).forEach(ciudad => {
        html += `<h4>${ciudad}</h4><canvas id="chart-${ciudad}" width="500"
height="250"></canvas>`;
      });
      olapDiv.innerHTML = html;
      // Dibujar los gráficos
      Object.keys(grouped).forEach(ciudad => {
        const canvas = document.getElementById(`chart-${ciudad}`);
        if (!canvas) return;
        const ctx = canvas.getContext('2d');
        // Eje X: Mes-Género, Eje Y: escuchas
        const labels = grouped[ciudad].map(item => `${item.month}\n${item.genero}`);
        const values = grouped[ciudad].map(item => item.listens);
        // Escalado
        const max = Math.max(...values, 1);
        const barWidth = Math.max(20, 400 / values.length);
        ctx.clearRect(0, 0, 500, 250);
```

```
        // Dibujar barras
        values.forEach((v, i) => {
          ctx.fillStyle = '#4a90e2';
          ctx.fillRect(40 + i * barWidth, 200 - (v / max) * 180, barWidth - 4, (v / max) *
180);
          // Mostrar valor encima de la barra
          ctx.fillStyle = '#333';
          ctx.font = '10px Arial';
          ctx.fillText(v, 40 + i * barWidth, 195 - (v / max) * 180);
        });
        // Dibujar etiquetas de género y mes debajo de cada barra, sin rotar y con salto de
línea
        labels.forEach((label, i) => {
          const [mes, genero] = label.split('\n');
          ctx.save();
          ctx.font = '10px Arial';
          ctx.textAlign = 'center';
          ctx.fillStyle = '#222';
          ctx.fillText(mes, 40 + i * barWidth + (barWidth - 4) / 2, 215);
          ctx.fillStyle = '#e26a4a';
          ctx.font = 'bold 11px Arial';
          ctx.fillText(genero, 40 + i * barWidth + (barWidth - 4) / 2, 230);
          ctx.restore();
        });
        // Ejes
        ctx.beginPath();
        ctx.moveTo(35, 200);
        ctx.lineTo(480, 200);
        ctx.moveTo(40, 200);
        ctx.lineTo(40, 20);
        ctx.strokeStyle = '#000';
        ctx.stroke();
        ctx.font = '12px Arial';
        ctx.fillText('Escuchas', 5, 30);
      });
    }
  } catch (err) {
    olapDiv.innerHTML = `<p>Error al conectar al backend.</p>`;
  }
});

// --- Insertar Usuario ---
// Evento para el formulario de nuevo usuario
document.getElementById('insert-usuario-form').addEventListener('submit', async (e) => {
  e.preventDefault(); // Evita recarga
  const usuario_id = document.getElementById('nuevo_usuario_id').value; // ID usuario
  const nombre = document.getElementById('nuevo_usuario_nombre').value; // Nombre
  const ciudad = document.getElementById('nuevo_usuario_ciudad').value; // Ciudad
  const resultDiv = document.getElementById('insert-usuario-result'); // Div resultado
  resultDiv.innerHTML = 'Insertando...'; // Mensaje de carga
  try {
    // Envía los datos al backend por POST
    const response = await fetch(`${backendUrl}/usuario`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ usuario_id, nombre, ciudad })
    });
    const data = await response.json(); // Respuesta JSON
    if (data.error) {
```

```
      // Muestra error si ocurre
      resultDiv.innerHTML = `<p style='color:red;'>Error: ${data.error}</p>`;
    } else {
      // Muestra éxito
      resultDiv.innerHTML = `<p style='color:green;'>${data.mensaje}</p>`;
    }
  } catch (err) {
    // Error de conexión
    resultDiv.innerHTML = `<p style='color:red;'>Error al conectar al backend.</p>`;
  }
});

// --- Insertar Canción ---
// Evento para el formulario de nueva canción
document.getElementById('insert-cancion-form').addEventListener('submit', async (e) => {
  e.preventDefault(); // Evita recarga
  const cancion_id = document.getElementById('nueva_cancion_id').value; // ID canción
  const titulo = document.getElementById('nueva_cancion_titulo').value; // Título
  const artista = document.getElementById('nueva_cancion_artista').value; // Artista
  const genero = document.getElementById('nueva_cancion_genero').value; // Género
  const resultDiv = document.getElementById('insert-cancion-result'); // Div resultado
  resultDiv.innerHTML = 'Insertando...'; // Mensaje de carga
  try {
    // Envía los datos al backend por POST
    const response = await fetch(`${backendUrl}/cancion`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ cancion_id, titulo, artista, genero })
    });
    const data = await response.json(); // Respuesta JSON
    if (data.error) {
      // Muestra error si ocurre
      resultDiv.innerHTML = `<p style='color:red;'>Error: ${data.error}</p>`;
    } else {
      // Muestra éxito
      resultDiv.innerHTML = `<p style='color:green;'>${data.mensaje}</p>`;
    }
  } catch (err) {
    // Error de conexión
    resultDiv.innerHTML = `<p style='color:red;'>Error al conectar al backend.</p>`;
  }
});

// --- Insertar Escucha ---
// Evento para el formulario de nueva escucha
document.getElementById('insert-escucha-form').addEventListener('submit', async (e) => {
  e.preventDefault(); // Evita recarga
  const usuario_id = document.getElementById('nueva_escucha_usuario_id').value; // ID usuario
  const cancion_id = document.getElementById('nueva_escucha_cancion_id').value; // ID canción
  const fecha_escucha = document.getElementById('nueva_escucha_fecha').value; // Fecha
  const resultDiv = document.getElementById('insert-escucha-result'); // Div resultado
  resultDiv.innerHTML = 'Insertando...'; // Mensaje de carga
  try {
    // Envía los datos al backend por POST
    const response = await fetch(`${backendUrl}/escucha`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ usuario_id, cancion_id, fecha_escucha })
    });
```

```
    const data = await response.json(); // Respuesta JSON
    if (data.error) {
      // Muestra error si ocurre
      resultDiv.innerHTML = `<p style='color:red;'>Error: ${data.error}</p>`;
    } else {
      // Muestra éxito
      resultDiv.innerHTML = `<p style='color:green;'>${data.mensaje}</p>`;
    }
  } catch (err) {
    // Error de conexión
    resultDiv.innerHTML = `<p style='color:red;'>Error al conectar al backend.</p>`;
  }
});

// --- Carga masiva de CSV ---
// Evento para el formulario de carga de CSV
document.getElementById('csv-upload-form').addEventListener('submit', async (e) => {
  e.preventDefault(); // Evita recarga
  const tipo = document.getElementById('csv-type').value; // Tipo de datos a cargar
  const fileInput = document.getElementById('csv-file'); // Input de archivo
  const resultDiv = document.getElementById('csv-upload-result'); // Div resultado
  if (!fileInput.files.length) {
    // Si no hay archivo seleccionado, muestra error
    resultDiv.innerHTML = '<p style="color:red;">Selecciona un archivo CSV.</p>';
    return;
  }
  const file = fileInput.files[0]; // Archivo seleccionado
  const formData = new FormData();
  formData.append('csv', file); // Agrega el archivo al formData
  try {
    // Envía el archivo al backend por POST
    const response = await fetch(`${backendUrl}/cargar-csv/${tipo}`, {
      method: 'POST',
      body: formData
    });
    const data = await response.json(); // Respuesta JSON
    if (data.error) {
      // Muestra error si ocurre
      resultDiv.innerHTML = `<p style='color:red;'>Error: ${data.error}</p>`;
    } else {
      // Muestra éxito
      resultDiv.innerHTML = `<p style='color:green;'>${data.mensaje}</p>`;
    }
  } catch (err) {
    // Error de conexión
    resultDiv.innerHTML = `<p style='color:red;'>Error al conectar al backend.</p>`;
  }
});
```

index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Sistema de Recomendación de Música</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Sistema de Recomendación de Música</h1>

    <section id="recommendation-section">
      <h2>Recomendación Personalizada</h2>
      <form id="recommendation-form">
        <label for="usuario_id">Ingresa tu ID de usuario:</label>
        <input type="number" id="usuario_id" name="usuario_id" required>
        <label for="modo_recomendacion">Modo de recomendación:</label>
        <select id="modo_recomendacion" name="modo_recomendacion">
          <option value="ciudad">Por ciudad</option>
          <option value="genero">Por género favorito</option>
        </select>
        <button type="submit">Obtener Recomendaciones</button>
      </form>
      <div id="recommendation-result"></div>
    </section>

    <section id="olap-section">
      <h2>Análisis OLAP: Escuchas por Género, Mes y Ciudad</h2>
      <button id="fetch-olap">Ver OLAP 2D</button>
      <div id="olap-result"></div>
    </section>

    <section id="insert-section">
      <h2>Insertar Usuario, Canción o Escucha</h2>
      <div style="display: flex; gap: 30px; flex-wrap: wrap; justify-content: center;">
        <form id="insert-usuario-form">
          <h3>Nuevo Usuario</h3>
          <input type="number" id="nuevo_usuario_id" placeholder="ID Usuario" required><br>
          <input type="text" id="nuevo_usuario_nombre" placeholder="Nombre" required><br>
          <input type="text" id="nuevo_usuario_ciudad" placeholder="Ciudad" required><br>
          <button type="submit">Insertar Usuario</button>
          <div id="insert-usuario-result"></div>
        </form>
        <form id="insert-cancion-form">
          <h3>Nueva Canción</h3>
          <input type="number" id="nueva_cancion_id" placeholder="ID Canción" required><br>
          <input type="text" id="nueva_cancion_titulo" placeholder="Título" required><br>
          <input type="text" id="nueva_cancion_artista" placeholder="Artista" required><br>
          <input type="text" id="nueva_cancion_genero" placeholder="Género" required><br>
          <button type="submit">Insertar Canción</button>
          <div id="insert-cancion-result"></div>
        </form>
        <form id="insert-escucha-form">
          <h3>Nueva Escucha</h3>
          <input type="number" id="nueva_escucha_usuario_id" placeholder="ID Usuario"
required><br>
```

```
          <input type="number" id="nueva_escucha_cancion_id" placeholder="ID Canción"
required><br>
          <input type="date" id="nueva_escucha_fecha" required><br>
          <button type="submit">Insertar Escucha</button>
          <div id="insert-escucha-result"></div>
        </form>
      </div>
    </section>

    <section id="csv-upload-section">
      <h2>Cargar datos masivos desde CSV</h2>
      <form id="csv-upload-form">
        <label for="csv-type">Tipo de datos:</label>
        <select id="csv-type" required>
          <option value="usuarios">Usuarios</option>
          <option value="canciones">Canciones</option>
          <option value="escuchas">Escuchas</option>
        </select>
        <input type="file" id="csv-file" accept=".csv" required>
        <button type="submit">Cargar CSV</button>
      </form>
      <div id="csv-upload-result"></div>
    </section>
  </div>

  <script src="app.js"></script>
</body>
</html>
```

Docker-compose.yml

```
version: '3.8'

services:
  cassandra:
    image: cassandra:4.1
    container_name: cassandra
    ports:
      - "9042:9042"
    environment:
      - CASSANDRA_START_RPC=true
    volumes:
      - cassandra_data:/var/lib/cassandra
    healthcheck:
      test: ["CMD", "cqlsh", "-e", "describe keyspaces"]
      interval: 30s
      timeout: 10s
      retries: 10

volumes:
  cassandra_data:
```

Archivos de información:

Funcion-ario: Explicación detallada del proyecto

Descripción General

Este sistema es una aplicación web de recomendación de música que utiliza una base de datos NoSQL (Cassandra) y permite:

- Recomendar canciones personalizadas a los usuarios.
- Analizar escuchas por género y mes (OLAP simplificado).
- Insertar usuarios, canciones y escuchas desde la web.
- Cargar datos masivos desde archivos CSV.

Estructura del Proyecto

- **docker-compose.yml**: Levanta Cassandra en un contenedor Docker.
- **backend/**: Código del servidor Node.js.
  - **index.js**: Lógica principal del backend y endpoints.
  - **cassandra-setup.js**: Configura el keyspace y las tablas en Cassandra.
  - **import_data.js**: Importa datos desde CSV a Cassandra.
  - **data/**: Archivos CSV de usuarios, canciones y escuchas.
- **frontend/**: Aplicación web (HTML, CSS, JS).
  - **index.html**: Interfaz de usuario.
  - **app.js**: Lógica de interacción con el backend.
  - **styles.css**: Estilos visuales.
- **docs/**: Documentación y diccionario de funciones.

Backend (Node.js + Express)

- **Conexión a Cassandra**: Usa el driver oficial, crea el keyspace y las tablas si no existen.
- **Endpoints principales**:
  - `/recommendation`: Devuelve recomendaciones personalizadas según el usuario y el modo (por ciudad o por género favorito).
  - `/olap/genre-month`: Devuelve un análisis OLAP de escuchas agrupadas por género y mes.
  - `/usuario`, `/cancion`, `/escucha`: Permiten insertar usuarios, canciones y escuchas vía POST.

- `/cargar-csv/:tipo`: Permite cargar datos masivos desde archivos CSV (usuarios, canciones o escuchas).

- **import_data.js**: Script para importar los datos iniciales desde los CSV a Cassandra.

Frontend (HTML + JS + CSS)

- **index.html**: Contiene formularios para:

  - Obtener recomendaciones personalizadas (seleccionando modo).

  - Ver análisis OLAP.

  - Insertar usuarios, canciones y escuchas.

  - Cargar archivos CSV masivos.

- **app.js**: Maneja los eventos de los formularios, realiza peticiones al backend y muestra los resultados.

- **styles.css**: Da formato visual moderno y responsivo a la web.

Flujo de Uso

1. ****Despliegue**:** Se levanta Cassandra con Docker y el backend con Node.js.

2. ****Carga de datos**:** Se pueden importar datos iniciales desde CSV o cargar nuevos desde la web.

3. ****Interacción**:** El usuario puede:

   - Consultar recomendaciones personalizadas.

   - Ver análisis OLAP.

   - Insertar nuevos datos manualmente o por CSV.

Lógica de Recomendación

- **Por ciudad**: Recomienda las canciones más escuchadas por usuarios de la misma ciudad.

- **Por género favorito**: Recomienda las canciones más escuchadas del género que más escucha el usuario.

Capturas de Pantalla de la base de datos

A continuación, se muestran algunas capturas de pantalla tablas que componen la base de datos:

1. **Tabla de usuarios**

```
cqlsh:musicrec> SELECT * FROM usuarios;

 usuario_id | ciudad       | nombre
------------+--------------+--------------------
        114 |      Caracas |       Diego Torres
        110 |        Quito |      Andrés Castro
        128 | Buenos Aires |    Ignacio Paredes
        117 |     Santiago |       Daniela Soto
        144 |      Caracas |      Matías Guzmán
        120 |        Quito |      Esteban Silva
        140 |        Quito |     Lucas Carrasco
        129 |   Montevideo |  Florencia Cabrera
        132 |       Bogota |    Leandro Aguirre
        185 |       Bogota |    Milagros Zamora
        105 |       Bogota |     Sofia Martinez
        123 |    Mexico DF |      Renata Castro
        153 |    Mexico DF |    Martina Pizarro
        178 | Buenos Aires |       Ignacio León
        137 |     Santiago |    Constanza Bravo
        118 | Buenos Aires |        Manuel Peña
        194 |      Caracas |       Matías López
        125 |       Bogota |      Julieta Gómez
        197 |     Santiago |    Juliana Salazar
        134 |      Caracas |        Bruno Godoy
        139 |   Montevideo |   Martina Figueroa
        169 |   Montevideo |   Victoria Herrera
        145 |       Bogota |    Agustina Duarte
        111 |      Caracas |     Valentina Ríos
        113 |    Mexico DF |      Paula Morales
        160 |        Quito |      Emilia Castro
        151 |      Caracas |      Solana Roldán
          1 |      Caicara |            Mansour
        157 |     Santiago |       Agustín Vera
        130 |        Quito |      Samuel Ortega
        122 |       Bogota |     Matías Herrera
        176 |         Lima |        Tomás Ponce
        177 |     Santiago |    Antonia Cordero
```

2. **Tabla de escuchas**

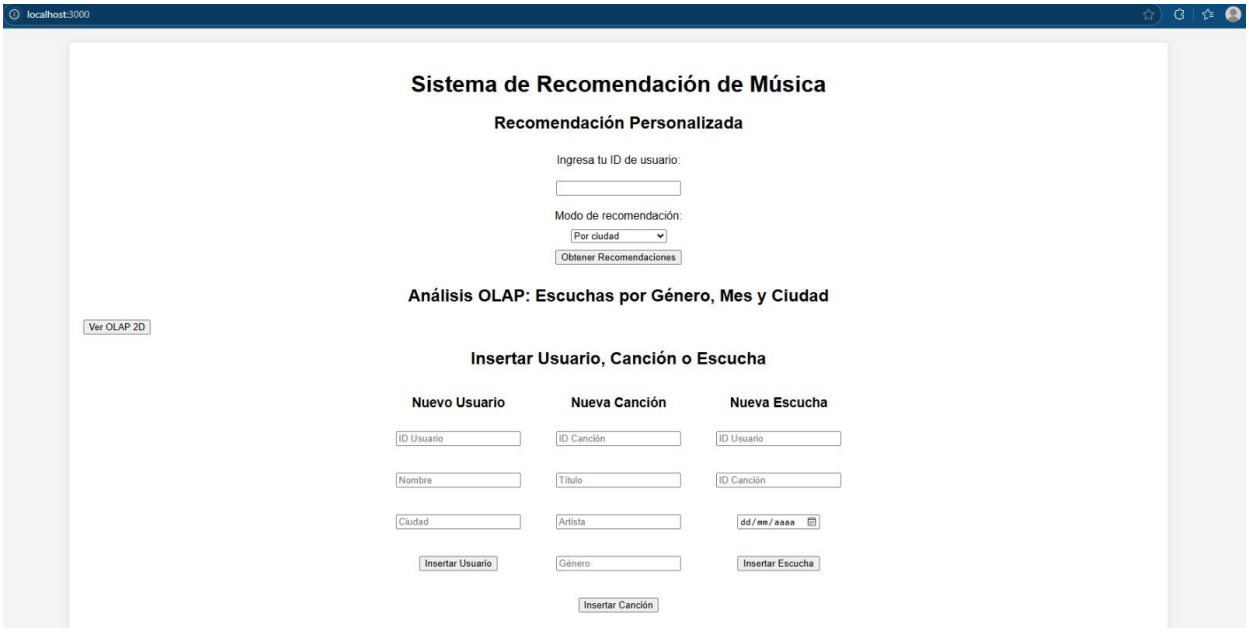
```
cqlsh:musicrec> SELECT * FROM escuchas;

 usuario_id | fecha_escucha | cancion_id
------------+---------------+------------
        114 |    2024-02-09 |         14
        110 |    2024-02-05 |         10
        128 |    2024-02-23 |         28
        117 |    2024-02-12 |         17
        144 |    2024-03-11 |         44
        120 |    2024-02-15 |         20
        140 |    2024-03-07 |         40
        129 |    2024-02-24 |         29
        132 |    2024-02-27 |         32
        185 |    2024-04-21 |         85
        105 |    2024-01-19 |          5
        123 |    2024-02-18 |         23
        153 |    2024-03-20 |         53
        178 |    2024-04-14 |         78
        137 |    2024-03-04 |         37
        118 |    2024-02-13 |         18
        194 |    2024-04-30 |         94
        125 |    2024-02-20 |         25
        197 |    2024-05-03 |         97
        134 |    2024-03-01 |         34
        139 |    2024-03-06 |         39
        169 |    2024-04-05 |         69
        145 |    2024-03-12 |         45
        111 |    2024-02-06 |         11
        113 |    2024-02-08 |         13
        160 |    2024-03-27 |         60
        151 |    2024-03-18 |         51
          1 |    2025-06-04 |          4
        157 |    2024-03-24 |         57
        130 |    2024-02-25 |         30
        122 |    2024-02-17 |         22
        176 |    2024-04-12 |         76
        177 |    2024-04-13 |         77
```

3. **Tabla de canciones**

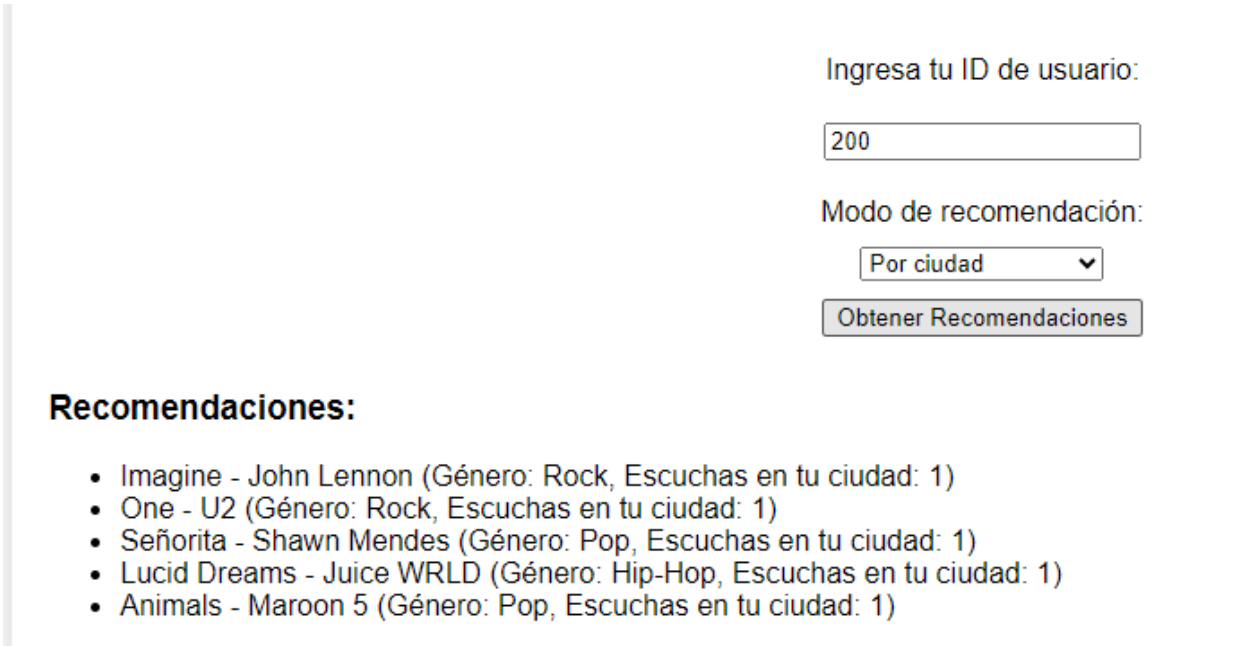
```
cqlsh:musicrec> SELECT * FROM canciones
           ... ;

 cancion_id | artista          | genero    | titulo
------------+------------------+-----------+-------------------------
         23 |         Dua Lipa |       Pop |              Levitating
         53 |     Justin Bieber |       Pop |           Love Yourself
         91 |            Ke$ha |       Pop |                  Praying
         55 |     Justin Bieber |       Pop |              Intentions
         33 |            Adele |       Pop |                   Hello
          5 |   Michael Jackson |       Pop |             Billie Jean
         28 |        Lil Nas X |   Country |           Old Town Road
         42 |            Drake |   Hip-Hop |         In My Feelings
         50 |         Maroon 5 |       Pop |                 Animals
         95 |    Jennifer Lopez |       Pop |            On The Floor
         88 |      Clean Bandit |       Pop |              Rather Be
         82 |      Major Lazer |       EDM |                Lean On
         49 |         Maroon 5 |       Pop |                  Sugar
         10 |      John Lennon |      Rock |                Imagine
         83 |      Major Lazer |       EDM |           Light It Up
         60 |         Coldplay |      Rock |              Paradise
         67 |      OneRepublic |       Pop |              Apologize
         79 |     Martin Garrix |       EDM |                Ocean
         16 |       Ed Sheeran |       Pop |              Perfect
         63 |         Coldplay |      Rock |               Fix You
         80 |     Martin Garrix |       EDM |          High on Life
         13 |            Oasis |      Rock |            Wonderwall
         30 |      Shawn Mendes |       Pop |             Señorita
         72 |           Avicii |       EDM |               Levels
         99 |    Jennifer Lopez |       Pop | Love Don't Cost a Thing
         11 |   Michael Jackson |       Pop |              Thriller
          1 |            Queen |      Rock |      Bohemian Rhapsody
         19 |       Ed Sheeran |       Pop |      Thinking Out Loud
         61 |         Coldplay |      Rock |                Yellow
         46 | The Chainsmokers |       Pop |                 Paris
         43 |            Drake |   Hip-Hop |          Hotline Bling
```

Pruebas de la aplicación:

Pagina inicial



Muestra recomendación por ciudad



Muestra recomendación por genero

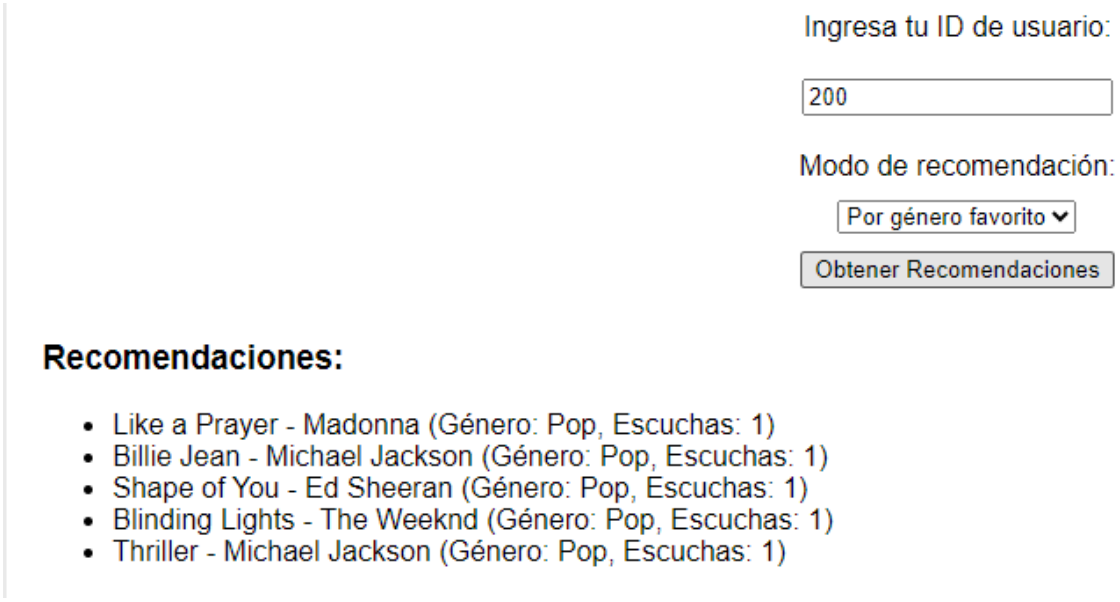


Grafico OLAP

Zona de inserción de datos: