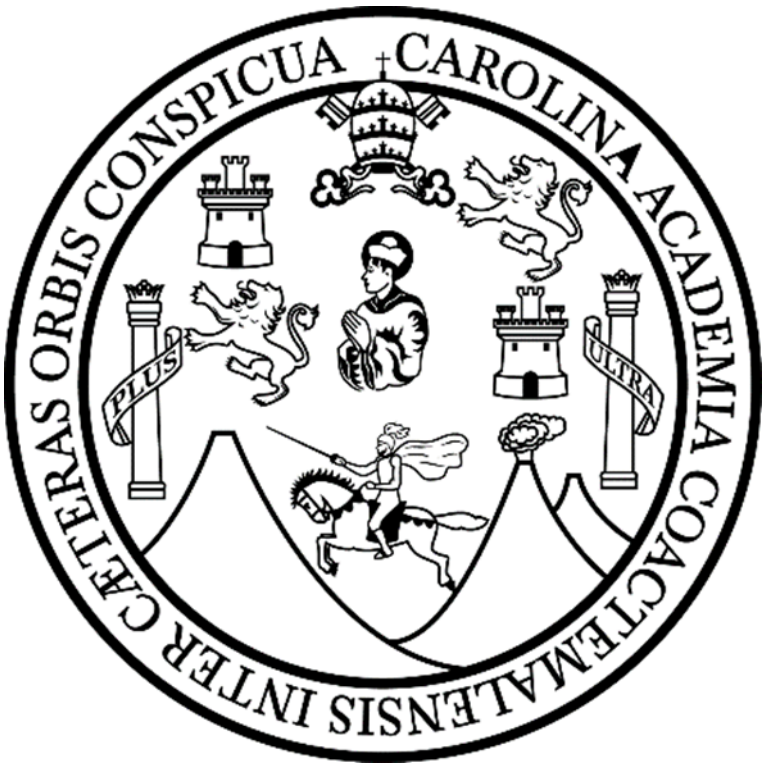
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Facultad de Ingeniería
Escuela de Ciencias y Sistemas



Proyecto 1 Fase 2

Sistema de Bases de Datos 2 “B”
GRUPO No. 2

NOMBRE	CARNET
Juan Francisco Urbina Silva	201906051
Alexis Marco Tulio López Cacoj	201908359
Camilo Ernesto Sincal Sipac	202000605

Introducción

En los sistemas de datos de gran escala, la migración de información entre bases de datos heterogéneas es una tarea crítica, que demanda un enfoque estructurado y recursos significativos para asegurar la integridad y accesibilidad de la información en un nuevo entorno. Los archivos *artista.js*, *canciones.js* y *discografia.js, los cuales* se han diseñado específicamente para migrar datos entre PostgreSQL, una base de datos relacional, y MongoDB, una base de datos NoSQL orientada a documentos. Estos scripts implementan un proceso meticuloso de extracción, transformación y carga (ETL) para trasladar de manera desnormalizada una gran variedad de datos musicales, incluidos detalles de artistas, canciones, discografías, géneros, etiquetas, lanzamientos y otras características asociadas.

Cada archivo JavaScript maneja un tipo específico de información y realiza la migración en lotes para gestionar eficientemente grandes volúmenes de datos. Sin embargo, debido a la naturaleza intensiva en recursos de estos procesos, el rendimiento y la velocidad de migración pueden variar considerablemente. Factores como la capacidad de procesamiento, la memoria RAM disponible y la velocidad del disco duro del equipo en uso influyen directamente en la duración de la migración. En entornos con recursos limitados, el proceso puede ser notablemente más lento, especialmente al migrar datos extensos y complejos.

Estos scripts también incluyen funciones auxiliares para enriquecer los datos trasladados a MongoDB, adaptándose al formato de documentos y agrupando información relacionada que en PostgreSQL está distribuida en varias tablas. Este diseño aporta flexibilidad y un enfoque desnormalizado a la base de datos de destino, optimizando su estructura para operaciones de consulta que podrían ser costosas en un entorno relacional. Sin embargo, la transformación de los datos relacionales a un formato de documentos también añade una carga extra al proceso, especialmente cuando se manejan relaciones complejas o datos anidados, como los de estos sistemas de música.

El enfoque general de esta migración se basa en un proceso por lotes que equilibra la eficiencia y la integridad de los datos. No obstante, la complejidad de los datos y la estructura altamente relacional de la base de datos hacen que este proceso sea sensible a las limitaciones de hardware, subrayando la importancia de contar con un entorno de ejecución robusto para minimizar tiempos de inactividad y demoras en la migración.

Objetivos

Los objetivos de los archivos artista.js, canciones.js y discografia.js en el proceso de migración de datos entre PostgreSQL y MongoDB se centran en:

1. **Migración Desnormalizada de Datos:** Convertir datos relacionales altamente normalizados en PostgreSQL a una estructura de documentos en MongoDB. Esto facilita el acceso y la consulta de datos en MongoDB mediante un enfoque de documentos anidados y estructurados que es más eficiente para ciertas consultas.
2. **Preservación de Integridad y Enriquecimiento de Datos:** Asegurar que toda la información relevante (artistas, canciones, discografías, lanzamientos, etiquetas, etc.) se migre completamente y se organice de manera coherente en la base de datos de destino. Este proceso incluye funciones auxiliares que transforman y enriquecen los datos, como la adición de detalles de lanzamiento, etiquetas y canciones para cada entidad relevante.
3. **Optimización del Rendimiento:** Realizar la migración en lotes configurables para administrar de manera eficiente el consumo de memoria y mejorar el rendimiento en la migración de grandes volúmenes de datos. Los lotes permiten procesar fragmentos manejables de datos, reduciendo la carga en los sistemas con recursos limitados.
4. **Flexibilidad en la Consulta:** Crear una base de datos final en MongoDB que esté optimizada para consultas rápidas y eficientes en una estructura desnormalizada. Esta estructura mejora las consultas en MongoDB, especialmente para aplicaciones donde se requiere un acceso rápido a los datos anidados y relacionados.
5. **Manejo de Escalabilidad y Recursos del Sistema:** Ajustar el proceso de migración para ser ejecutable en equipos con diferentes capacidades de hardware, con la conciencia de que los tiempos de migración variarán en función de los recursos disponibles.

Contenido

Artistas (index.js)

Descripción

Este archivo es un script de migración que extrae información sobre artistas, géneros, lugares, álbumes y canciones desde una base de datos PostgreSQL y la inserta en una base de datos MongoDB. Este proceso se realiza en lotes para optimizar la migración de un volumen grande de datos, con una serie de funciones auxiliares para extraer, transformar y enriquecer los datos antes de insertarlos.

Código

Importación de Módulos

```
const { MongoClient } = require('mongodb');
const { Client } = require('pg');
const { performance } = require('perf_hooks');
```

- MongoClient: Se utiliza para la conexión e interacción con MongoDB.
- Client: Permite la conexión e interacción con PostgreSQL.
- performance: Se usa para medir el tiempo de ejecución de cada lote de migración.

Función Principal migrate

Esta función se encarga de conectar ambas bases de datos, configurar los parámetros de migración por lotes y coordinar el proceso de migración.

```
async function migrate() { ... }
```

1. Conexión a MongoDB

```
const mongoClient = new MongoClient('mongodb://localhost:27017', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
await mongoClient.connect();
const db = mongoClient.db('mongoMusicBrainz');
```

Establece una conexión con la base de datos MongoDB en localhost y selecciona la base de datos mongoMusicBrainz.

2. Conexión a PostgreSQL

```
const pgClient = new Client({
    user: 'postgres',
  host: 'localhost',
  database: 'fase1',
  password: 'abc123**',
  port: 5432,
});
await pgClient.connect();
```

Conecta a PostgreSQL con las credenciales y base de datos especificadas.

3. Tamaño de Lote

```
const batchSize = 10000;
```

Define el tamaño de cada lote para la migración, lo cual ayuda a controlar el consumo de memoria y tiempos de respuesta.

Funciones Auxiliares de Extracción de Datos

Cada función auxiliar se encarga de extraer datos específicos desde PostgreSQL.

- getGenresForArtists

Obtiene los géneros asociados a un conjunto de artistas.

```
async function getGenresForArtists(artistIds) { ... }
```

Recibe un arreglo de IDs de artistas y devuelve un mapa de géneros asociados por ID de artista.

- getTagsForArtists

Obtiene los tags (etiquetas) asociados a un conjunto de artistas.

```
async function getTagsForArtists(artistIds) { ... }
```

Similar a getGenresForArtists, pero devuelve un mapa de tags por artista.

- getPlacesForArtists

Extrae y asocia los lugares relevantes (inicio, final, y lugar de origen) para los artistas.

```
async function getPlacesForArtists(artistRows) { ... }
```

Recibe filas de artistas y devuelve un mapa de nombres de lugares.

- getSongsForAlbums

Recupera las canciones de un conjunto de álbumes.

```
async function getSongsForAlbums(albumIds) { ... }
```

Devuelve un mapa de canciones agrupadas por ID de álbum, incluyendo detalles de cada canción.

- getAlbumsForArtists

Obtiene los álbumes de un conjunto de artistas, incluyendo información de canciones asociadas.

```
async function getAlbumsForArtists(artistIds) { ... }
```

Devuelve un mapa de álbumes con sus canciones asociadas, agrupados por ID de artista.

Función migrateInBatches

Esta función organiza el proceso de migración en lotes, limitando el número de filas migradas en cada iteración.

```
async function migrateInBatches(query, mongoCollection) { ... }
```

- Parámetros:
  - query: Consulta SQL base para seleccionar los datos de artistas.
  - mongoCollection: Colección de MongoDB en la que se insertarán los datos.
- Flujo:
  - Inicialización: Define el offset inicial y marca hasMoreRows como true.
  - Bucle de Lotes: Ejecuta la consulta SQL para obtener un conjunto de artistas.
  - Extracción de Datos Asociados: Llama a las funciones auxiliares para enriquecer los datos con tags, álbumes, canciones y lugares.
  - Transformación y Envío a MongoDB: Mapea los datos de artistas y los inserta en MongoDB.
  - Medición de Tiempo: Calcula y muestra en consola el tiempo de migración para cada lote.

Ejecución de la Migración Completa

```
await migrateInBatches(
  `SELECT a.*, t.nombre AS tipo_nombre, g.genero AS genero_nombre, l.nombre AS lugar_nombre
   FROM Artista a
   LEFT JOIN TipoArtista t ON a.tipo = t.id
   LEFT JOIN Genero g ON a.genero = g.id
   LEFT JOIN lugar l ON a.id_lugar = l.id`,
  db.collection('artistas')
);
```

Este comando realiza la migración, invocando migrateInBatches con una consulta SQL que selecciona datos enriquecidos de los artistas y especificando la colección de artistas en MongoDB como destino.

Cierre de Conexiones

```
await pgClient.end();
await mongoClient.close();
```

Cierra las conexiones a PostgreSQL y MongoDB al finalizar la migración.

Ejecución de la Función migrate

```
migrate().catch(console.error);
```

Inicia el proceso de migración y captura cualquier error que ocurra durante la ejecución, mostrando un mensaje de error en consola.

Canciones (canciones.js)

Descripción General

Este archivo es un script de migración de datos que transfiere información de canciones y lanzamientos desde una base de datos PostgreSQL a una base de datos MongoDB. Extrae detalles sobre las canciones, los lanzamientos asociados, los países y etiquetas de lanzamiento, y los inserta en MongoDB de manera desnormalizada. El proceso de migración se realiza en lotes para manejar grandes volúmenes de datos de forma eficiente.

Código

Importación de Módulos

```
const { MongoClient } = require('mongodb');
const { Client } = require('pg');
const { performance } = require('perf_hooks');
```

- MongoClient: Permite la conexión e interacción con MongoDB.
- Client: Facilita la conexión e interacción con PostgreSQL.
- performance: Se usa para medir el tiempo de ejecución de cada lote de migración.

Función Principal migrate

La función migrate se encarga de establecer las conexiones con ambas bases de datos, definir el tamaño de los lotes, y coordinar el proceso de migración.

```
async function migrate() { ... }
```

Conexión a MongoDB

```
const mongoClient = new MongoClient('mongodb://localhost:27017', {
    useNewUrlParser: true,
  useUnifiedTopology: true,
});
await mongoClient.connect();
const db = mongoClient.db('mongoMusicBrainz');
```

Establece una conexión con MongoDB y selecciona la base de datos mongoMusicBrainz.

Conexión a PostgreSQL

```
const pgClient = new Client({
  user: 'postgres',
  host: '172.17.144.27',
  database: 'musicbrainz',
  password: 'kmilo0289',
  port: 5432,
});
await pgClient.connect();
```

Conecta a PostgreSQL en el servidor y base de datos especificados.

Tamaño de Lote

```
const batchSize = 10000;
```

Define el tamaño de cada lote para la migración, controlando así el rendimiento y consumo de memoria.

Funciones Auxiliares de Extracción de Datos

Cada función auxiliar extrae datos específicos de PostgreSQL y los transforma en el formato necesario para MongoDB.

- getDataPaises

Obtiene información de los países (lugares) donde se lanzó cada álbum.

```
async function getDataPaises(lanzamientoIds) { ... }
```

Recibe un arreglo de IDs de lanzamientos y devuelve un mapa de lugares con fechas asociadas por lanzamiento.

- getTagLanzamiento

Obtiene las etiquetas (tags) asociadas a cada lanzamiento.

```
async function getTagLanzamiento(lanzamientoId) { ... }
```

Devuelve un mapa de tags asociados a cada lanzamiento.

- getLanzamientoData

Extrae la información de lanzamientos, incluyendo lugar, idioma, catálogo y etiquetas.

```
async function getLanzamientoData(trackIds) { ... }
```

- Llama a las funciones getDataPaises y getTagLanzamiento para enriquecer los datos de lanzamientos.
- Devuelve un mapa de lanzamientos con los datos enriquecidos, agrupado por ID de track.

Función migrateInBatches

Esta función organiza el proceso de migración en lotes, limitando el número de registros migrados en cada iteración.

```
async function migrateInBatches(query, mongoCollection) { ... }
```

- Parámetros:
  - query: Consulta SQL base para seleccionar los datos de canciones.
  - mongoCollection: Colección de MongoDB en la que se insertarán los datos.
- Flujo:
  - Inicialización: Define el offset inicial y marca hasMoreRows como true.
  - Bucle de Lotes: Ejecuta la consulta SQL para obtener un conjunto de canciones.
  - Extracción de Datos Asociados: Llama a getLanzamientoData para obtener información de lanzamientos enriquecida.
  - Transformación y Envío a MongoDB: Mapea los datos de canciones y los inserta en MongoDB.
  - Medición de Tiempo: Calcula y muestra en consola el tiempo de migración para cada lote.

Ejecución de la Migración Completa

```
await migrateInBatches(
  `select distinct t.id, t.nombre, m.formato as formato_cancion, l.id as id_lanzamiento, a.nombre as nombre_artista, t.duracion
   from track t
   inner join artistacredit a on a.id_artista_credito = t.id_creditos
   inner join medio m on t.id_medio = m.id
   inner join lanzamientos l on m.release_id = l.id`,
  db.collection('canciones')
);
```

Este comando ejecuta la migración llamando a migrateInBatches con una consulta SQL para seleccionar datos de canciones, y especifica canciones como la colección de destino en MongoDB.

Cierre de Conexiones

```
await pgClient.end();
await mongoClient.close();
```

Cierra las conexiones a PostgreSQL y MongoDB al finalizar la migración.

Ejecución de la Función migrate

```
migrate().catch(console.error);
```

Inicia el proceso de migración y captura cualquier error que ocurra durante la ejecución, mostrando un mensaje de error en consola.

Discografías (discografia.js)

Descripción General

Este archivo es un script de migración que transfiere datos de discografías y sus detalles asociados (canciones, etiquetas, sellos discográficos y lanzamientos) desde una base de datos PostgreSQL a una base de datos MongoDB. La migración se realiza en lotes para optimizar el manejo de grandes volúmenes de datos y mejorar la eficiencia.

Código

Importación de Módulos

```
const { MongoClient } = require('mongodb');
const { Client } = require('pg');
const { performance } = require('perf_hooks');
```

- MongoClient: Permite la conexión e interacción con MongoDB.
- Client: Facilita la conexión e interacción con PostgreSQL.
- performance: Mide el tiempo de ejecución de cada lote de migración.

Función Principal migrate

La función migrate establece las conexiones con ambas bases de datos, configura el tamaño del lote y coordina la migración.

```
async function migrate() { ... }
```

- Conexión a MongoDB

```
const mongoClient = new MongoClient('mongodb://localhost:27017', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
await mongoClient.connect();
const db = mongoClient.db('mongoMusicBrainz');
```

Conecta a MongoDB y selecciona la base de datos mongoMusicBrainz.

- Conexión a PostgreSQL

```
const pgClient = new Client({
  user: 'postgres',
  host: 'localhost',
  database: 'proyecto',
  password: 'super',
  port: 5432,
});
await pgClient.connect();
```

Conecta a PostgreSQL en localhost utilizando las credenciales y base de datos proporcionadas.

- Tamaño de Lote

```
const batchSize = 10000;
```

Define el tamaño del lote para controlar el rendimiento y consumo de memoria durante la migración.

Funciones Auxiliares de Extracción de Datos

Cada función auxiliar extrae y organiza información específica de PostgreSQL.

- getAlbumsDetails

Obtiene los detalles de álbumes específicos como el nombre del artista, tipo, título, fecha de salida y número de lanzamientos.

```
async function getAlbumsDetails(albumIds) { ... }
```

Recibe un arreglo de IDs de álbumes y devuelve un mapa de detalles de álbumes, donde la clave es el ID del álbum.

- getSongsForAlbums

Obtiene las canciones asociadas a un conjunto de álbumes.

```
async function getSongsForAlbums(albumIds) { ... }
```

Devuelve un mapa de canciones por ID de álbum, con detalles de cada canción como nombre y duración.

- getTagsForAlbums

Extrae las etiquetas (tags) asociadas a un conjunto de álbumes.

```
async function getTagsForAlbums(albumIds) { ... }
```

Devuelve un mapa de tags por ID de álbum.

- getLabelsForAlbums

Obtiene información de sellos discográficos (labels) asociados a un conjunto de álbumes, incluyendo código de barras y año de lanzamiento.

```
async function getLabelsForAlbums(albumIds) { ... }
```

Devuelve un mapa de etiquetas de lanzamiento por ID de álbum.

Función migrateInBatches

Esta función organiza el proceso de migración en lotes, limitando el número de registros migrados en cada iteración.

```
async function migrateInBatches(query, mongoCollection) { ... }
```

- Parámetros:
  - query: Consulta SQL base para seleccionar los datos de discografía.
  - mongoCollection: Colección de MongoDB donde se insertarán los datos.
- Flujo:
  - Inicialización: Define offset y marca hasMoreRows como true.
  - Bucle de Lotes: Ejecuta la consulta SQL para obtener datos de artistas relacionados con discografías.
  - Extracción de Datos Asociados: Obtiene los detalles de álbumes, canciones, etiquetas y sellos discográficos a través de funciones auxiliares.
  - Transformación y Inserción: Estructura y organiza los datos en formato JSON y los inserta en MongoDB.
  - Medición de Tiempo: Calcula y muestra el tiempo de migración para cada lote en consola.

Ejecución de la Migración Completa

```
await migrateInBatches(
  `SELECT a.*, t.nombre AS tipo_nombre, g.genero AS genero_nombre, l.nombre AS lugar_nombre
   FROM Artista a
   LEFT JOIN TipoArtista t ON a.tipo = t.id
   LEFT JOIN Genero g ON a.genero = g.id
   LEFT JOIN lugar l ON a.id_lugar = l.id`,
  db.collection('discografias')
);
```

Este comando inicia la migración, ejecutando migrateInBatches con una consulta SQL que selecciona datos enriquecidos de artistas relacionados con discografía, y especifica discografias como la colección de destino en MongoDB.

Cierre de Conexiones

```
await pgClient.end();
await mongoClient.close();
```

Cierra las conexiones a PostgreSQL y MongoDB al finalizar la migración.

Ejecución de la Función migrate

```
migrate().catch(console.error);
```

Inicia el proceso de migración y captura cualquier error que ocurra durante la ejecución, mostrando un mensaje de error en consola.

Conclusiones

- La migración de datos relacionales a una base de datos NoSQL como MongoDB presenta desafíos técnicos y de rendimiento, especialmente al convertir una estructura relacional en una estructura desnormalizada. Al utilizar scripts de JavaScript diseñados para ejecutar un proceso de extracción, transformación y carga (ETL) en lotes, se logra manejar eficazmente grandes volúmenes de datos mientras se preserva la integridad de la información y se optimiza su formato para MongoDB.
- El uso de funciones auxiliares en estos scripts permite enriquecer los datos de origen, adaptándolos al modelo de documentos de MongoDB y agrupando elementos relacionados, como etiquetas, canciones, detalles de lanzamiento y datos de artistas. Este proceso, aunque intensivo en recursos, crea una base de datos final que favorece consultas rápidas y directas en aplicaciones que requieren acceso eficiente a datos multidimensionales.
- La dependencia de recursos de hardware en esta migración subraya la necesidad de ajustar cuidadosamente el tamaño de los lotes y tener en cuenta las capacidades del sistema en uso, ya que equipos con limitaciones de procesamiento o memoria experimentarán tiempos de migración más largos. Con equipos robustos, sin embargo, estos scripts logran una migración completa y estructurada que maximiza la usabilidad y flexibilidad de la base de datos en MongoDB, logrando el objetivo de una migración eficaz y organizada que mantiene la riqueza de los datos originales en un formato optimizado para consulta.