Generación de BBDD de metadatos de Música desestructurada con versionado de documentos en MongoDB

Manuel Rodríguez y Felipe Castellanos
Bases de Datos No Relacionales 2023
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay

Resumen

En este artículo se presenta un proyecto de migración del sistema de carga de datos de una base de datos con metadatos de una aplicación de streaming de música. Actualmente, la carga de estos metadatos se realiza mediante un sistema desarrollado con Pentaho PDI, el cual lee archivos depositados en un bucket de AWS S3, los procesa y los almacena en una base de datos MySQL. Sin embargo, el crecimiento excesivo de las tablas de la base de datos MySQL ha comenzado a complicar la tarea de realizar cambios en la estructura para soportar los nuevos requerimientos de las versiones más recientes de DDEX. Para abordar este problema, se propone un nuevo sistema de carga desarrollado en Python que requiere menos procesamiento.

ÍNDICE

I. Introducción

El presente artículo tiene como objetivo proponer una estrategia de migración para el sistema de carga de metadatos de una aplicación de streaming de música. Actualmente, el sistema utiliza Pentaho PDI para procesar y almacenar los datos en una base de datos MySQL desde archivos del estándar DDEX que son depositados en un bucket de AWS S3. Sin embargo, debido al crecimiento exponencial de las tablas en la base de datos MySQL y la complejidad para adaptarse a las nuevas versiones de DDEX, se hace necesario replantear la estrategia de carga y almacenamiento de los datos.

Para resolver estos desafíos, se propone la implementación de un nuevo sistema de carga en Python que convierta los documentos DDEX en formato JSON y los almacene en una colección de MongoDB. De esta forma, se evita la fragmentación de archivos y las actualizaciones de múltiples tablas en MySQL, lo que permitirá una carga más eficiente y escalable.

Adicionalmente, se empleará el patrón "schema versioning pattern"para gestionar las consultas a la base de datos MongoDB. Este enfoque permitirá abstraer la versión de DDEX en la cual están estructurados los documentos, facilitando la consulta de datos para las diferentes versiones del estándar.

El artículo se organiza de la siguiente manera: En la sección II, se detallará la propuesta de migración, incluyendo los requerimientos, el análisis y el diseño de la solución. La sección III abarcará la Implementación, donde se detallará el script de carga, luego en la sección IV detallaremos los Patrones de Diseño, teniendo uno utilizado y el otro identificado como mejora a lo implementado. La siguiente sección (V Experimentación y Consultas) detallará las consultas realizadas y una breve explicación de las mismas, para terminar con VI Conclusiones y trabajo a futuro, donde se presentarán las conclusiones obtenidas del trabajo realizado, las posibles extensiones a la estrategia propuesta y las sugerencias para mejoras a realizar.

II. Propuesta de Migración

II-A. Requerimientos y criterios de evaluación de la migración

II-A1. Requerimientos de la migración: La migración del sistema de carga de datos de metadatos requiere abordar varios requerimientos específicos para garantizar una transición exitosa hacia la nueva estrategia. A continuación, se describen los principales requerimientos que deben ser considerados:

II-A1a. Compatibilidad con el estándar DDEX: Durante la migración, es fundamental asegurar la compatibilidad con el estándar DDEX. Esto implica garantizar que las estructuras de datos, los formatos de archivos y las reglas de validación definidas por DDEX se mantengan intactas en la nueva solución basada en MongoDB. Además, se debe asegurar la correcta interpretación de los elementos de metadatos, como el título de la canción, el nombre del artista, la duración, el género, entre otros. Asimismo, se requiere validar los archivos según las especificaciones de DDEX para asegurar la integridad de los datos cargados en la base de datos MongoDB.

II-A1b. Rendimiento y eficiencia: En el contexto de la migración del sistema de carga de metadatos, se establecen requisitos de rendimiento y eficiencia. Se espera que la migración sea capaz de manejar grandes volúmenes de datos de manera eficiente, garantizando un tiempo máximo permitido para el procesamiento de archivos. Por ejemplo, se espera que la migración pueda procesar y almacenar los metadatos de un archivo DDEX de tamaño promedio (aproximadamente 1 MB) en menos de 1 segundo. Esto asegurará una respuesta rápida y una experiencia fluida para los usuarios de la aplicación de streaming de música.

II-A1c. Escalabilidad: Uno de los requisitos clave para la migración del sistema de carga de metadatos es garantizar la escalabilidad de la solución. Se espera que la nueva solución basada en MongoDB pueda manejar el crecimiento futuro de la cantidad de archivos de metadatos sin experimentar degradación en el rendimiento. Por ejemplo, se espera que la migración pueda adaptarse sin problemas a un aumento del 50 % en la carga de archivos de metadatos mensuales durante el próximo año, sin afectar negativamente los tiempos de procesamiento ni el tiempo de respuesta de las consultas.

II-A1d. Facilidad de mantenimiento: Otro aspecto importante en la migración del sistema de carga de datos de metadatos es abordar los requerimientos relacionados con la facilidad de mantenimiento de la nueva solución. Se espera que la implementación en Python y MongoDB permita realizar cambios y actualizaciones de manera ágil y eficiente, evitando interrupciones significativas en el servicio. Para lograr esto, se buscará utilizar una arquitectura modular y bien documentada, que facilite la identificación y corrección de problemas, así como la incorporación de nuevas funcionalidades sin afectar el funcionamiento general del sistema. Además, se contemplará la disponibilidad de herramientas que permitan el monitoreo y la administración proactiva de la base de datos MongoDB, con el fin de anticipar y resolver posibles inconvenientes antes de que afecten la experiencia del usuario.

II-A1e. Independencia de la versión de DDEX: La migración del sistema de carga de datos de metadatos debe garantizar la capacidad de la aplicación de música para realizar consultas de manera independiente de la versión específica del documento JSON de DDEX en el que estén estructurados los metadatos. Para lograr esto, se requerirá la implementación del 'schema versioning pattern' en la nueva solución. Este patrón permitirá que la aplicación consulte los metadatos sin estar acoplada a una versión específica, lo que facilitará la incorporación de actualizaciones futuras en el estándar DDEX sin afectar el funcionamiento de la aplicación. Además, se deberá establecer un mecanismo para identificar la versión del documento DDEX asociado a cada registro de metadatos en MongoDB, lo que permitirá una correcta interpretación de los datos durante la consulta. Al aplicar el 'schema versioning pattern', se asegurará la compatibilidad y la adaptabilidad de la aplicación de música a medida que evolucione el estándar DDEX.

II-A2. Criterios de evaluación: Con el fin de evaluar el éxito de la migración del sistema de carga de datos de metadatos, se establecen los siguientes criterios de evaluación, los cuales permitirán determinar si se han cumplido los requerimientos establecidos:

II-A2a. Eficiencia en el acceso a los datos: La eficiencia en el acceso a los datos en la nueva base de datos MongoDB será evaluada mediante la comparación de los tiempos de respuesta de consultas comunes realizadas en MySQL y en MongoDB. Se seleccionarán consultas representativas que reflejen los diferentes tipos de consultas utilizadas en la aplicación de música, como la búsqueda de canciones por género, la obtención de información de un artista específico o la recuperación de listas de reproducción. Se ejecutarán estas consultas tanto en la base de datos MySQL existente como en la nueva base de datos MongoDB y se medirán los tiempos de respuesta para cada una de ellas. Se establecerá un criterio de comparación, por ejemplo, un objetivo de mejora del 30 % en los tiempos de respuesta en MongoDB en comparación con MySQL. El cumplimiento de este objetivo indicará que la migración ha logrado una mejora significativa en la eficiencia del acceso a los datos.

II-A2b. Escalabilidad de la solución: La escalabilidad de la solución será evaluada considerando aspectos como el rendimiento y la capacidad de manejar un aumento significativo en la carga de datos y en el número de consultas concurrentes. Se realizarán pruebas de carga simulando escenarios de alta demanda, donde se incrementará gradualmente la cantidad de datos y el número de usuarios concurrentes. Durante estas pruebas, se medirá el tiempo de respuesta de las consultas y se evaluará si la solución mantiene un rendimiento aceptable sin degradación significativa a medida que aumenta la carga. Por ejemplo, se podría establecer un criterio de evaluación que indique que la solución debe ser capaz de procesar un aumento del 50 % en la carga de datos y mantener un tiempo de respuesta promedio por debajo de X segundos para un número determinado de consultas concurrentes. Cumplir con estos criterios demostrará que la solución es escalable y puede adaptarse eficientemente a un crecimiento futuro en la carga de datos y en la demanda de consultas concurrentes.

II-B. Análisis de la solución actual

II-B1. Descripción del sistema actual: El sistema actual de carga de metadatos se basa en la herramienta de integración de datos Pentaho PDI, en servicios de AWS y utiliza una base de datos MySQL para el almacenamiento.

El proceso de carga comienza con la recepción de archivos en formato DDEX, los cuales son depositados por múltiples distribuidores de música diferentes en la nube de Amazon Web Services (AWS) S3. Estos archivos son ingresados a una base de datos MySQL llamada "releases", la cual luego será utilizada para en sus múltiples tablas almacenar los datos sin procesar del archivo DDEX, que se almacenarán sin procesar pero si ya fraccionados dependiendo de las diferentes secciones que existen en los archivos DDEX. Puede entenderse a la base releases como una transformación "casi directa" de la estructura de un archivo DDEX en XML a tablas de una base de datos relacional MySQL.

A través de Pentaho PDI, un primer subproceso llamado "Process Releases" se realiza una lectura sobre la tabla 'releases.s3 delivery queue' para obtener la referencia a los archivos ingresados en los buckets y se lleva a cabo la extracción de los DDEX desde AWS S3, seguido de un proceso de transformación "casi directa" para adaptar los datos al esquema de la base de datos releases en MySQL. Posteriormente y a través de otro subproceso llamado "Load to Catalog", los metadatos son leídos desde la base releases, procesados, cargados en tablas específicas dentro de la base de datos "catalog" de MySQL. Esta base de datos catalog si ya es la base de datos final, la cual tiene el catálogo de música ya ordenado y mantiene cierto nivel de normalización. Es también aquí donde consulta directamente el API utilizado por la aplicación de streaming de producción.

Luego que los recursos son procesados y están en el formato estándar, otro subproceso de Pentaho PDI llamado "Copy Resources to S3 Catalog" copia estos recursos ya procesados al un bucket llamado catalog que si ya es el bucket final, desde el cual la aplicación de streaming descarga las canciones utilizando una estrategia similar a un stream mientras realiza la reproducción en el dispositivo del usuario final.

Otros dos subprocesos de Pentaho PDI participan en el final del proceso, uno llamado "Activate Messages" se encarga de chequear que el release haya sido procesado correctamente y lo activa en la base "catalog". El otro, llamado "Send ACKs"

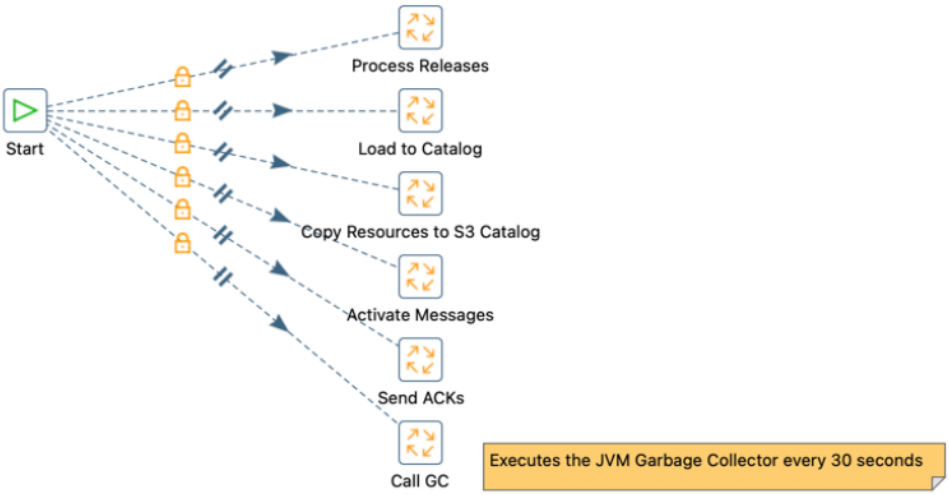

Figura 1: Proceso ejecutado en Pentaho DI

envía al distribuidor del release lo que se conoce ACK, que consiste en un reporte con los detalles del procesamiento del release, típicamente si el procesamiento fue exitoso o si hubo errores con los detalles correspondientes al caso.

Los componentes principales del sistema de carga incluyen:

- AWS S3 Bucket: Es el repositorio donde se reciben y almacenan los archivos DDEX que contienen los metadatos de la música.
- Pentaho PDI: Esta herramienta se encarga de leer los archivos desde el bucket de S3, aplicar las transformaciones necesarias, realizar la carga en la base de datos MySQL y enviar a AWS BATCH las URLs de AWS S3 de los archivos de audio a convertir.
- Base de datos MySQL: Es el sistema de gestión de bases de datos relacional utilizado para almacenar los metadatos procesados. En este motor se encuentran 2 bases de datos, la base “releases” que tiene los datos con el mapeo “casi directo” del esquema de los archivos y la base “catalog” que almacena los datos ya procesados y normalizados para ser consumidos por el API de la aplicación de streaming.
- AWS BATCH: Aquí se encuentran configuradas múltiples instancias de máquinas AWS EC2 Linux que realizan la conversión de miles de archivos por minuto de música digitalizada en formatos mp3, wav, flac, etc.
- AWS Lambda: Esta función está escuchando eventos en el bucket de S3 y mantiene una conexión abierta a la base ’releases’ en MySQL. Cuando un archivo nuevo es depositado en el bucket, inserta un registro en la tabla releases.s3 delivery queue”para que el sistema de carga en el Pentaho PDI se entere de que ese archivo fue depositado y que está listo para ser procesado.

Estos componentes trabajan en conjunto para llevar a cabo el flujo de carga de datos desde la recepción de archivos hasta su almacenamiento en la base de datos MySQL, permitiendo que la aplicación de streaming de música tenga acceso a la información relevante para su funcionamiento.

La Figura 1 ilustra la raíz del proceso que se ejecuta en un Pentaho PDI. La misma corresponde a un Pentaho Job, el cual ejecuta en paralelo los subprocesos ya mencionados en esta sección, que también son Jobs que se ejecutan iterativamente para realizar sus correspondientes tareas.

II-B2. Funcionalidades y características: Este sistema actual de carga de metadatos, basado en Pentaho PDI y MySQL, cuenta con una serie de funcionalidades y características clave que lo hacen a día de hoy lo suficientemente efectivo en el procesamiento de los archivos DDEX pero, se estima que en un futuro próximo esto no vaya a cubrir las necesidades al 100 %. Estas características son:

1. Procesamiento de diferentes tipos de archivos DDEX: El sistema es capaz de procesar una amplia variedad de archivos DDEX, que contienen metadatos de diferentes canciones, álbumes y artistas. Esto incluye archivos de diferentes versiones del estándar DDEX.

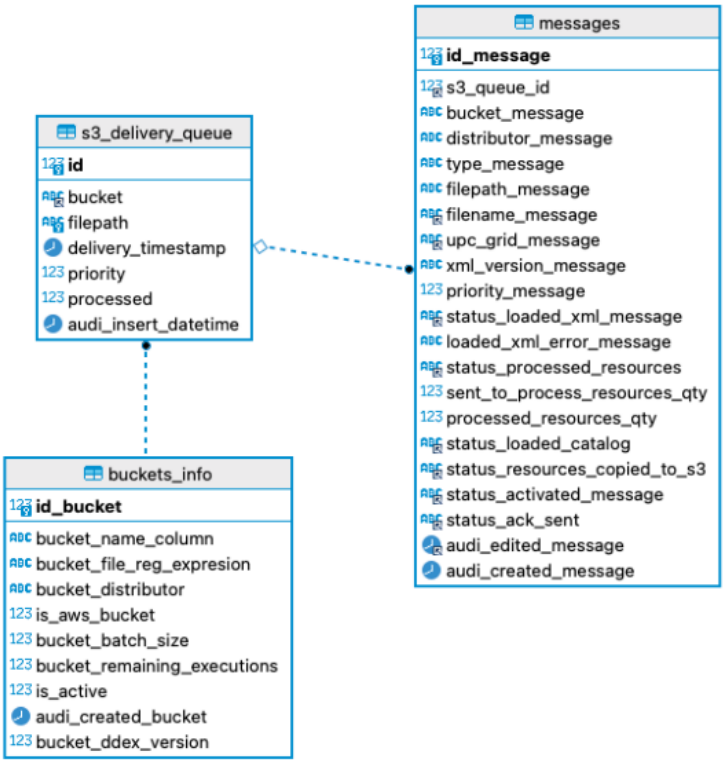

Figura 2: Tablas de configuración del entorno actual

2. Transformación de datos para su almacenamiento en MySQL: Mediante el uso de transformaciones en Pentaho PDI, los datos extraídos de los archivos DDEX se adaptan y se transforman al formato adecuado para su almacenamiento en la base de datos MySQL. Esta transformación asegura la consistencia y la integridad de los datos.
3. Gestión de la actualización de registros existentes: El sistema tiene la capacidad de identificar registros existentes en la base de datos MySQL y gestionar su actualización. Esto permite mantener la información actualizada y reflejar los cambios realizados en los metadatos de las canciones.

Esta arquitectura general del sistema es orquestada a través de configuración en un conjunto de 3 tablas configuradas en la base de datos denominada “releases”. Esta configuración ordena el procesamiento de los datos, indicando a los subprocesos en Pentaho PDI las tareas a realizar. En la Figura 2 se presenta un pequeño diagrama de este conjunto de 3 tablas donde están configurados los datos para la orquestación.

Cabe destacar que esta solución ha demostrado su capacidad para procesar un volumen considerable de releases por día, en el rango de 60,000 a 70,000. Al día de hoy, la capacidad del sistema para manejar grandes volúmenes de datos de manera eficiente, lo que es crucial para el funcionamiento de la aplicación de streaming de música.

II-B3. Limitaciones y desafíos: Sin embargo, a medida que el sistema ha evolucionado y las nuevas versiones del estándar DDEX han sido lanzadas, se han identificado importantes limitaciones y desafíos que afectan su escalabilidad y mantenimiento.

Una de las principales limitaciones radica en la estructura de la base de datos en MySQL, donde las tablas han sido mapeadas directamente a los archivos de DDEX. Con el tiempo, el crecimiento exponencial de los datos ha ocasionado que la ejecución de cambios en la estructura de estas tablas se vuelva compleja y propensa a errores. La adaptación a las nuevas versiones del estándar DDEX requiere alteraciones frecuentes en la base de datos, lo que afecta el rendimiento y dificulta la implementación de mejoras.

Otra limitación significativa se encuentra en la base ’catalog’, que actúa como la base final del procesamiento y es consultada por la aplicación de música a través de su API. Esta base experimenta un aumento constante de datos debido a la gran cantidad de inserciones y actualizaciones que deben mantener la integridad de los datos en las tablas relacionadas. Como resultado, la actualización de registros en la base ’catalog’ se ha vuelto más lenta, lo que impacta directamente en la experiencia del usuario y en la disponibilidad de la información más actualizada.

Estas limitaciones y desafíos están obstaculizando la capacidad del sistema actual para mantenerse al día con las últimas versiones del estándar DDEX y para manejar eficientemente el crecimiento continuo de datos.

II-B4. Rendimiento y escalabilidad: La evaluación del rendimiento y la escalabilidad del sistema actual revela algunos desafíos importantes. Actualmente, el tiempo requerido para procesar los archivos de metadatos utilizando Pentaho PDI y MySQL es considerable, lo que afecta la eficiencia del proceso de carga de datos. y las consultas en MySQL.

En cuanto a la escalabilidad, se ha identificado que hacer escalar este sistema de manera tradicional resulta costoso. Aunque actualmente se utilizan dos máquinas en paralelo para ejecutar los procesos de Pentaho PDI, aumentar el número de máquinas se vuelve prohibitivo en términos de costos. Se requiere una solución más eficiente y rentable que permita manejar grandes volúmenes de datos sin incurrir en gastos excesivos.

Teniendo en cuenta estos desafíos, es crucial considerar una nueva solución que permita mejorar el rendimiento y la escalabilidad del sistema de carga de datos de metadatos. La migración a una plataforma de almacenamiento como MongoDB ofrece una alternativa más eficiente, brindando la posibilidad de manejar grandes volúmenes de datos y consultas concurrentes de manera más efectiva, sin incurrir en gastos excesivos y complejidades asociadas al escalado tradicional de MySQL. Esto permite además, abstraerse del problema del almacenamiento de los datos y centrarse como problema principal en las consultas a la base, siendo un problema en lugar de dos.

III. Implementación

III-A. Estrategia de Migración

III-A1. Diseño de la nueva estrategia de Migración: La estrategia de migración propuesta utiliza Python y MongoDB para la carga de datos de metadatos. El proceso comienza con la conexión a un bucket de Amazon S3, donde se almacenan los archivos de metadatos. Utilizando el cliente de Amazon S3 proporcionado por la biblioteca boto3, se realiza una búsqueda recursiva en la carpeta especificada para obtener todos los objetos que contienen los archivos XML.

Una vez obtenidos los objetos, se procede a procesarlos de manera individual. Se verifica la extensión del archivo para asegurarse de que sea un archivo XML válido. A continuación, se lee el contenido del archivo utilizando el método ‘get object‘ del cliente de Amazon S3, y se decodifica el contenido en formato UTF-8.

El siguiente paso es convertir el contenido XML en un diccionario de Python utilizando la biblioteca xmltodict. Posteriormente, se transforma el diccionario en formato JSON mediante la función ‘json.dumps()‘. Este JSON resultante contiene la representación estructurada de los datos de metadatos, la cual es perfectamente almacenable e indexable en una colección de MongoDB.

La estrategia de migración propuesta aprovecha las capacidades de Python y MongoDB para simplificar el proceso de carga de metadatos, transformándolos de archivos XML a documentos JSON y almacenadolos eficientemente en la colección de la base de datos MongoDB.

A continuación queda adjunto el código de Python utilizado para efectuar esta tarea, destacando su sencillez en comparación con el proceso de carga descrito anteriormente.

Listado 1 Script de Carga

```
import boto3
import xmltodict
import json
import pymongo
from pymongo.server_api import ServerApi

bucket_name = 'onerpm-delivery'
folder_path = 'received/'

s3 = boto3.client('s3')

# Usamos el paginador de boto3 para manejar grandes cantidades de objetos
paginator = s3.get_paginator('list_objects_v2')

# Configuramos los filtros para solo obtener los objetos que se encuentran en la carpeta especificada
prefix = folder_path
delimiter = '/'

# Recursivamente obtenemos todos los objetos en la carpeta especificada
response = s3.list_objects_v2(Bucket=bucket_name, Prefix=folder_path)

# Conectar a MongoDB
#uri = "mongodb+srv://pruebas:pruebas_bdnr@bdnrcluster0.tls286r.mongodb.net/?retryWrites=true&w=majority"
uri = "mongodb+srv://felipe:felipe@cluster0.wtlzqpd.mongodb.net/?retryWrites=true&w=majority"
client = pymongo.MongoClient(uri, server_api=ServerApi('1'))
db = client['releases']
collection = db['releases']

while True:
   # Recorremos cada objeto obtenido
   for obj in response.get('Contents', []):
      # Procesamos solo los objetos que tengan una extensin .xml
```

```
        if obj['Key'].lower().endswith('.xml'):
            # Obtenemos el contenido del archivo
            file_obj = s3.get_object(Bucket=bucket_name, Key=obj['Key'])
            file_content = file_obj['Body'].read().decode('utf-8')

            # Convertimos el contenido XML a un diccionario Python
            xml_dict = xmltodict.parse(file_content)

            # Convertimos el diccionario a JSON
            json_data = json.dumps(xml_dict)

            # Imprimimos el resultado
            #print(json_data)

            # Insertar documento en coleccin de MongoDB
            collection.insert_one(json.loads(json_data))

    # Si hay ms objetos, obtenemos la siguiente pgina
    try:
        token = response['NextContinuationToken']
        response = s3.list_objects_v2(Bucket=bucket_name, Prefix=folder_path, ContinuationToken=token)
    except KeyError:
        break
```

III-A2. Ventajas y desafíos de la nueva estrategia: La nueva estrategia de migración propuesta presenta una serie de ventajas y beneficios significativos en comparación con el sistema actual. En primer lugar, se destaca la simplicidad del proceso de carga de datos. Con la nueva estrategia, el archivo completo en formato JSON se carga directamente en la base de datos MongoDB, sin necesidad de particionarlo ni realizar actualizaciones o inserciones en múltiples tablas. Esto reduce considerablemente la complejidad del proceso y simplifica la gestión de los datos de metadatos.

Además de la simplicidad, la nueva estrategia ofrece una mayor flexibilidad y escalabilidad. Al utilizar MongoDB como base de datos, se obtiene una estructura de datos flexible que se adapta fácilmente a los cambios en los archivos DDEX y permite la incorporación de nuevas funcionalidades en el futuro. La implementación del schema versioning pattern en MongoDB facilita la actualización de los esquemas de datos a medida que evolucionan las versiones de DDEX, evitando la necesidad de realizar cambios en la estructura de la base de datos y minimizando el impacto en el sistema.

La capacidad de escalamiento horizontal de MongoDB permite manejar grandes volúmenes de datos y consultas concurrentes. Esto simplifica el mantenimiento del sistema al evitar cambios en la estructura de múltiples tablas de MySQL para adaptarse a nuevas versiones de DDEX. El uso del schema versioning pattern en MongoDB facilita la gestión de cambios de esquema y garantiza la integridad de los datos.

Como desafíos asociados a la nueva estrategia de carga se identifican los siguientes:

1. Mantener la sincronización y consistencia de datos entre la base de datos MySQL existente y la nueva base de datos MongoDB durante el período de transición.
2. Gestionar los cambios en los esquemas de datos debido a las actualizaciones de las versiones de DDEX utilizando el schema versioning pattern en MongoDB.
3. Adquirir los conocimientos necesarios sobre MongoDB y su configuración para garantizar un despliegue eficiente y óptimo.
4. Optimizar el rendimiento y escalabilidad de la base de datos MongoDB a medida que crecen los volúmenes de datos.
5. Establecer una estrategia de respaldo y recuperación para garantizar la seguridad y disponibilidad de los datos en MongoDB.

III-B. Estructura genérica de un archivo DDEX

Listado 2 Ejemplo de archivo DDEX

```
<?xml version="1.0" encoding="UTF-8"?>
<ern:ERNMessage xmlns:ern="http://ddex.net/xml/ern/xxx%22%3E
  <MessageHeader>
    <!-- Informacin de encabezado del mensaje -->
  </MessageHeader>
  <NewReleaseMessage>
    <MessageHeader>
      <!-- Informacin de encabezado especfica del mensaje de lanzamiento -->
    </MessageHeader>
    <MessageBody>
      <NewRelease>
        <!-- Informacin sobre el lanzamiento musical -->
      </NewRelease>
      <ResourceList>
        <!-- Lista de recursos asociados al lanzamiento -->
        <SoundRecording>
          <!-- Informacin sobre una grabacin de sonido especfica -->
        </SoundRecording>
        <!-- Otros recursos (como videos, letras, etc.) -->
```

```
      </ResourceList>
      <ReleaseList>
        <!-- Lista de lanzamientos relacionados -->
      </ReleaseList>
    </MessageBody>
  </NewReleaseMessage>
</ern:ERNMessage>
```

En el ejemplo, se presenta un documento XML que sigue el estándar DDEX (Digital Data Exchange), utilizado para la comunicación de información relacionada con la industria musical. El fragmento XML muestra la estructura básica de un mensaje ERN (Electronic Release Notification), utilizado para enviar información sobre lanzamientos musicales. Este es el estado en el que se encuentran los datos en su estado inicial previo a todo el tratamiento realizado para insertarlos en nuestra base de MongoDB.
A continuación se muestra el ejemplo:

- **ern:ERNMessage**: Tag raíz que engloba todo el mensaje DDEX.
- **xmlns:ern=”http://ddex.net/xml/ern/411”**: Representa la versión del archivo DDEX, la cual será utilizada para implementar el schema versioning pattern.
- **MessageHeader**: Contiene información de encabezado general del mensaje.
- Puede haber un **MessageHeader** tanto en el nivel del **ERNMessage** como en el nivel del **NewReleaseMessage**.
- **NewReleaseMessage**: Contiene información específica del mensaje de lanzamiento musical.
- **MessageBody**: Contiene el cuerpo del mensaje, incluyendo detalles del lanzamiento musical y los recursos asociados.
- **NewRelease**: Contiene información detallada sobre un lanzamiento musical específico.
- **ResourceList**: Lista de recursos asociados al lanzamiento (grabaciones de sonido, videos, letras, etc.).
- **SoundRecording**: Información específica sobre una grabación de sonido en particular.
- **ReleaseList**: Lista de lanzamientos relacionados (puede incluir lanzamientos anteriores o posteriores relacionados con el lanzamiento principal). Este y sus subtags son los más accedidos por consultas en búsqueda de datos de artistas, álbumes y canciones.
- **Otros tags adicionales**: Pueden existir otros tags específicos dentro de cada sección para proporcionar detalles más precisos sobre los encabezados, los lanzamientos y los recursos.

III-C. Implementación en este proyecto

Debido a las limitaciones de tiempo como tambien de recursos (se utilizó la base de datos de prueba de Atlas), no se escaló lo sufiente esta base de datos como se ilustra en lo teórico (aplicable a un ambiente productivo como lo es el caso de esta aplicación de música). Se cuenta con un volumen bajo de datos que no permite hacer pruebas de eficiencia a la hora de la carga de forma correcta (aunque se constata la diferencia en la velocidad de la carga en comparación con el método anterior). Esto no nos frena a poder contar con tres versiones distintas de los archivos DDEX, lo que nos permite explotar el patrón de diseño a utilizar (detallado en la siguiente sección) y mostrar la facilidad para poder agregar nuevas versiones, sin tener que editar nada en la carga y realizando modificaciones mínimas en las consultas.

A modo de resumen, podemos decir que este proyecto es un caso de uso para mostrar lo efectiva que puede llegar a ser esta solución en caso de contar con los recursos para poder realizarlo de forma completa.

IV. Patrones de diseño utilizados

IV-A. Schema Versioning Pattern

El gran problema que inspira a esta transformación de la estrategia a emplear para este caso de uso consiste en el manejo de las distintas versiones presentes dentro de los datos a cargar a la base de datos de la aplicación de música. Como mencionamos anteriormente, el formato DDEX recibe actualizaciones constantemente y eso repercute en las cargas a la base de datos relacional que las almacena, ya que hay que realizar un tratamiento distinto para cada una a la hora del almacenamiento.

El patrón de versionado de esquemas en MongoDB permite gestionar de manera efectiva estos cambios. Al cargar los datos directamente en MongoDB, se evita el sobreprocesamiento que podría ocurrir al tener que realizar modificaciones directas en una base de datos relacional.

La flexibilidad de MongoDB en cuanto al esquema permite adaptarse fácilmente a las diferentes versiones de los datos DDEX. A medida que se producen actualizaciones en el formato, se puede aplicar un tratamiento específico durante las consultas para manejar las diferencias en la estructura de los datos.

Al utilizar MongoDB, se puede aprovechar su capacidad para manejar datos no estructurados y semi estructurados, lo cual resulta beneficioso cuando se trabaja con diferentes versiones de los archivos DDEX.

En resumen, el uso del patrón de versionado de esquemas en MongoDB permite adaptarse de manera eficiente a los cambios en el formato DDEX, minimizando el impacto en la aplicación y asegurando la integridad y consistencia de los datos.

Se profundizará en el tratamiento de las distintas vesiones en la sección ’Experimentación-Consultas’.

IV-B. Subset pattern

Este patrón de diseño no esta implementado en esta solución pero luego abarcar este Proyecto queda claro que aplica perfectamente para estos requerimientos, dado que las consultas realizadas acceden siempre a la misma sección de un documento, mas especificamente en 'ReleaseList'.

En lugar de incorporar todos los datos relacionados en un solo documento, el 'subset pattern' divide los datos y almacena solo una parte de la información en cada documento, mientras que los campos adicionales se almacenan en otra colección.

Este enfoque se utiliza cuando se espera que los datos relacionados sean consultados o modificados por separado con cierta frecuencia, lo que puede mejorar el rendimiento y la escalabilidad.

En caso de que este proyecto termine desembarcando en un ambiente productivo, este patrón deberá ser empleado.

Se empleará quedandonos unicamente con el identificador, la versión y todo lo incluido en 'ReleaseList', siendo util para la mayoría de las consultas. Para las consultas que necesiten otras partes del documento original, se accederá al mismo pero será en los casos menos frecuentes.

V. Experimentación - Consultas

En la parte de Anexo podemos encontrar el Jupyter Notebook que cuenta con el codigo Python de las consultas realizadas y sus valores de retorno. Se puede apreciar distintas formas de abarcar el 'Schema Versioning Pattern' a la hora de realizar una consulta de datos no estructurados con el tratamiento de cada versión en particular. En todos los casos parametrizamos los valores de entrada para cada consulta, en el notebook se ve un valor por defecto, que nos permite desplegar resultados. El objetivo de esta sección es de mostrar distintos enfoques para consultar a esta base de datos de MongoDB generando un tratamiento por versión. Se intenta mostrar distintos enfoques y casos para abarcar esta problemática. En el marco de este proyecto se decide también agregar la versión, aunque no tiene valor en lo que respecta lo funcional.

Lo que se puede concluir de esta sección es la obtención de los valores de retorno deseado y la muestra de que se cuenta con una implementación que quizas no es extremadamente eficiente pero que permite el correcto manejo de las versiones como también el agregado de una nueva, ya que es agregar un caso mas al tratamiento ya realizado. Como valor adicional, al agregar una versión, el motor de base de datos continua funcionando correctamente durante el transcurso en el cual se está agregando la lógica para trabajar con esta versión, ignorando los nuevos campos pero continuando con un correcto funcionamiento.

V-A. Cantidad de releases por Versión

Listado 3 Cantidad de releases por Versión

```
    cantidad_de_releases = [
      {
         '$group': {
            '_id': '$ernm:NewReleaseMessage.@xmlns:ernm',
            'Cantidad_de_Releases': {
               '$sum': 1
            }
         }
      },
      {
         '$match': {
            '_id': {
               '$ne': None
            }
         }
      },
      {
         '$project': {
            'version': '$_id',
            'Cantidad_de_Releases': 1,
            '_id': 0
         }
      }
   ]
```

Este es el caso menos complejo de las queries realizadas, no estamos utilizando el 'Schema Versioning Pattern' mencionado anteriormente debido a que se busca también mostrar un caso en el cual el patrón de diseño no es necesario.

En resumen, este código de agregación en MongoDB agrupa los documentos por versión, cuenta la cantidad de releases en cada versión y devuelve un resultado que muestra la versión y la cantidad de releases correspondientes, excluyendo los documentos sin versión asignada (estos son los archivos que reflejan la completitud de los archivos DDEX).

V-B. Artistas por Album

Listado 4 Artistas por Album

```
album_id = '190295058418'
artistas_por_album = [
   {
      '$match': {
         'ernm:NewReleaseMessage.ReleaseList.Release.ReleaseId.ICPN.#text': album_id
      }
   },
   {
      '$project': {
         'artistas': {
            '$switch': {
               'branches': [
                  {
                     'case': {
                        '$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/411']
                     },
                     'then': '$ernm:NewReleaseMessage.ReleaseList.Release.DisplayArtistName.#text'
                  },
                  {
                     'case': {
                        '$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/382']
                     },
                     'then': {
                        '$arrayElemAt': ['$ernm:NewReleaseMessage.ReleaseList.Release.ReleaseDetailsByTerritory.
                            DisplayArtist.PartyName.FullName', 0]
                     }
                  },
                  {
                     'case': {
                        '$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/383']
                     },
                     'then': {
                        '$arrayElemAt': ['$ernm:NewReleaseMessage.ReleaseList.Release.ReleaseDetailsByTerritory.
                            DisplayArtist.PartyName.FullName', 0]
                     }
                  }
               ],
               'default': None
            }
         },
         'version': '$ernm:NewReleaseMessage.@xmlns:ernm'
      }
   },
   {
      '$unwind': '$artistas'
   }
]
```

Este pipeline de agregación en MongoDB filtra los documentos según un ID de álbum específico y luego realiza una proyección para seleccionar y transformar los campos relacionados con los artistas y las versiones. El resultado final es una colección de documentos con información sobre artistas y versiones asociadas al álbum proporcionado.

Mediante 'switch' efectuamos el tratamiento respectivo a cada versión, ya que contamos con distintas formas de acceder al campo deseado, el nombre del Artista de un lanzamiento musical.

En esta consulta podemos observar que las versiones http://ddex.net/xml/ern/382 y http://ddex.net/xml/ern/383 muestran un mismo comportamiento por lo que se necesita unicamente dos formas de efectuar esta consulta.

V-C. Tracks por Artista

Listado 5 Tracks por Artista

```
artist_name = "Turtles"
Tracks_por_Artista = [
   {
      '$match': {
         '$or': [
            {'ernm:NewReleaseMessage.ReleaseList.Release.DisplayArtistName.#text': artist_name},
            {'ernm:NewReleaseMessage.ReleaseList.Release.ReleaseDetailsByTerritory.DisplayArtist.PartyName.FullName':
                artist_name},
            {'ernm:NewReleaseMessage.ReleaseList.Release.ReleaseDetailsByTerritory.DisplayArtist.PartyName.FullName':
                artist_name}
         ]
      }
   },
   {
      '$facet': {
         'case_382_383': [
```

```
            {
               '$match': {
                  '$expr': {
                     '$or': [
                        {'$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/382']},
                        {'$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/383']}
                     ]
                  }
               }
            },
            {
               '$unwind': '$ernm:NewReleaseMessage.ReleaseList.Release'
            },
            {
               '$group': {
                  '_id': 0,
                  'tracks': {
                     '$push': '$ernm:NewReleaseMessage.ReleaseList.Release.ReferenceTitle.TitleText'
                  }
               }
            }
         ],
         'case411': [
            {
               '$match': {
                  '$expr': {
                     '$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/411']
                  }
               }
            },
            {
               '$unwind': '$ernm:NewReleaseMessage.ReleaseList.TrackRelease'
            },
            {
               '$group': {
                  '_id': 0,
                  'tracks': {
                     '$push': '$ernm:NewReleaseMessage.ReleaseList.TrackRelease.DisplayTitleText.#text'
                  }
               }
            }
         ]
      }
   },
   {
      '$project': {
         'tracks': {
            '$cond': [
               {'$gt': [{'$size': '$case_382_383.tracks'}, 0]},
               '$case_382_383.tracks',
               '$case411.tracks'
            ]
         },
         'version': '$ernm:NewReleaseMessage.@xmlns:ernm'
      }
   }
]
]
```

En este caso se filtra los documentos para encontrar aquellos relacionados con un artista específico (pasado por parametro) y luego obtiene los tracks de toda la colección asociados a ese artista. En caso de que una release sea un Album, se incluye todos los items incluidos. El resultado final es un arreglo de tracks y el nombre del artista en cuestión.

Lo interesante de este caso es el 'match' efectuado en primera instancia con las distintas posibilidades que tenemos de acceder al nombre del artista según la version, esto lo realizamos gracias a 'OR'. Luego al tener que incluir la lista entera de los tracks, optamos por 'facet' para poder implementar multiples etapas para cada versión. Esto genera una salida para cada versión y filtramos la que no sea vacía.

V-D. Tracks por Album

Listado 6 Tracks por Album

```
album_id = '190295058418'
tracks_por_album = [
   {
      '$match': {
         'ernm:NewReleaseMessage.ReleaseList.Release.ReleaseId.ICPN.#text': '190295058418'
      }
   },
   {
```

```
'$project': {
   'Album': {
      '$switch': {
         'branches': [
            {
               'case': {
                  '$or': [
                     {'$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/382']},
                     {'$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/383']}
                  ]
               },
               'then': {
                  '$cond': {
                     'if': {'$isArray': '$ernm:NewReleaseMessage.ReleaseList.Release'},
                     'then': {
                        '$arrayElemAt': [
                           '$ernm:NewReleaseMessage.ReleaseList.Release.ReferenceTitle.TitleText', 0
                        ]
                     },
                     'else': '$ernm:NewReleaseMessage.ReleaseList.Release.ReferenceTitle.TitleText'
                  }
               }
            },
            {
               'case': {
                  '$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/411']
               },
               'then': {
                  '$cond': {
                     'if': {'$isArray': '$ernm:NewReleaseMessage.ReleaseList.TrackRelease'},
                     'then': '$ernm:NewReleaseMessage.ReleaseList.Release.DisplayTitle.TitleText',
                     'else': '$ernm:NewReleaseMessage.ReleaseList.Release.DisplayTitle.TitleText'
                  }
               }
            }
         ],
         'default': None
      }
   },
   'Track_List': {
      '$switch': {
         'branches': [
            {
               'case': {
                  '$or': [
                     {'$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/382']},
                     {'$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/383']}
                  ]
               },
               'then': {
                  '$cond': {
                     'if': {'$isArray': '$ernm:NewReleaseMessage.ReleaseList.Release'},
                     'then': {
                        '$map': {
                           'input': '$ernm:NewReleaseMessage.ReleaseList.Release',
                           'as': 'release',
                           'in': '$$release.ReferenceTitle.TitleText'
                        }
                     },
                     'else': None
                  }
               }
            },
            {
               'case': {
                  '$eq': ['$ernm:NewReleaseMessage.@xmlns:ernm', 'http://ddex.net/xml/ern/411']
               },
               'then': {
                  '$cond': {
                     'if': {'$isArray': '$ernm:NewReleaseMessage.ReleaseList.TrackRelease'},
                     'then': {
                        '$map': {
                           'input': '$ernm:NewReleaseMessage.ReleaseList.TrackRelease',
                           'as': 'track',
                           'in': '$$track.DisplayTitleText.#text'
                        }
                     },
                     'else': '$ernm:NewReleaseMessage.ReleaseList.TrackRelease.DisplayTitleText.#text'
                  }
               }
            }
         ],
         'default': None
      }
   },
   'version': '$ernm:NewReleaseMessage.@xmlns:ernm'
```

```
            }
        }
    ]
```

En este caso, luego de filtrar por el identificador del album ingresado por el usuario, se busca devolver el titulo del mismo y todos sus elementos asociados. Se realiza una proyección para devolver estos elementos. En este caso como se desea retornar tanto el nombre del album, como todos los nombres de los elementos del mismo, hay que efectuar dos veces el tratamiento por versiones debido a que ambos campos cuentan con distintas representaciones para cada versión. Eso explica lo largo del codigo en comparación con 'Artistas de un Album', que cuentan con una logica similar pero distintos retornos.

VI. Conclusiones y trabajo futuro

En conclusión, la implementación de la nueva estrategia de migración utilizando Python y MongoDB para la carga de datos de metadatos basados en DDEX ofrece diversas ventajas y beneficios en comparación con el sistema actual. La simplicidad de la carga de archivos completos en formato JSON en la base de datos MongoDB, sin la necesidad de particionarlos y actualizar o insertar en múltiples tablas, reduce la complejidad del proceso y agiliza la carga de datos.

La adopción del schema versioning pattern permite abordar los desafíos asociados con los cambios en las versiones de DDEX, facilitando la actualización y adaptación de las consultas de los datos sin afectar significativamente la funcionalidad del sistema.

La utilización de MongoDB como base de datos ofrece un mayor rendimiento en comparación con MySQL para este caso, permitiendo manejar grandes volúmenes de datos y consultas concurrentes de manera más eficiente.

Sin embargo, es importante destacar que la implementación de la nueva estrategia también presenta desafíos. La migración de la base de datos actual a MongoDB requiere un proceso cuidadoso para garantizar la integridad y consistencia de los datos. Otro desafío posible es, a la hora de enfrentar la aplicación real de este caso de uso a un ambiente productivo, enfrentar la escalabilidad con grandes volumenes de datos y de carga como fue mencionado anteriormente.

En cuanto al trabajo futuro, se plantea la posibilidad de explorar técnicas de optimización y escalabilidad adicionales, como la implementación de técnicas de caching, sharding y la utilización de clústeres MongoDB para manejar grandes volúmenes de datos y consultas. Además, es necesaria la integración con herramientas de visualización y análisis de datos para completar el ciclo.

Sumado a lo mencionado anteriormente, se deberá implementar el patrón de diseño 'Subset Pattern' mencionado en secciones anteriores, ya que, al escalar el tamaño de esta implementación se volvería crucial a la hora de generar consultas óptimas.

En resumen, la nueva estrategia de migración utilizando Python y MongoDB representa una mejora significativa en términos de rendimiento, flexibilidad y facilidad de mantenimiento. Aunque se presentan desafíos, el potencial de escalabilidad y la capacidad de adaptarse a los cambios futuros hacen de esta estrategia una opción prometedora para el procesamiento y almacenamiento de metadatos de música.

VII. Bibliografía

- DDEX: (https://www.ddex.net/)
- AWS S3: (https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html)
- AWS Lambda: (https://docs.aws.amazon.com/lambda/latest/dg/welcome.html)
- AWS Batch: (https://docs.aws.amazon.com/batch/latest/userguide/what-is-batch.html)
- Pentaho PDI: (https://help.hitachivantara.com/Documentation/Pentaho/9.2/Products/Pentaho_Data_Integration?ecid=ms_glo_bd_en_sscepen01)
- MongoDB: (https://www.mongodb.com/docs/)
- Schema Versioning Pattern: (https://www.mongodb.com/blog/post/building-with-patterns-the-schema-versioning-pattern)
- Subset Pattern: (https://www.mongodb.com/blog/post/building-with-patterns-the-subset-pattern)

VIII. Anexo

A continuación dejamos el link del proyecto para poder visualizar el Notebook con las ejecuciones realizadas y sus resultados en formato Latex. El archivo se llama 'ProyectoBDNRnotebook.tex'.

Enlace: https://www.overleaf.com/read/fhxwfhfzwdrj