Proyecto 1(fase 2) MusicBrainz

Estudiante	Carné No.
Gerhard Benjamin Ardon Valdez	202004796
Eduardo Andres Cuevas Tzun	202004703
Sebastian Alejandro Velasquez Bonilla	202006635

Guatemala 29 de Octubre de 2024

Descripción

MusicBrainz es una base de datos abierta y colaborativa que pretende almacenar y organizar datos de música, artistas y recordings a lo largo de la historia de la industria musical. Su objetivo es proporcionar una fuente de datos precisa y libre sobre canciones, álbumes, artistas, sellos discográficos, fechas de lanzamiento y más.

Para este proyecto se utilizó una API en Go para transferir los datos de nuestra base de datos en PostgreSQL a MongoDB. Esto incluye la implementación de procedimientos almacenados, la extracción de datos, y la administración de bases de datos, como la revisión de collations, vacuums y la creación de backups completos y comprimidos.

Replicación de Base de Datos a MongoDB utilizando Go

La replicación de una base de datos a MongoDB utilizando Go es un proceso que permite transferir datos desde una base de datos relacional, como PostgreSQL la cual se utilizó para la práctica uno y se tenían cargados todos los datos hacia una base de datos NoSQL, en este caso MongoDB. Este enfoque aprovecha las capacidades de Go para manejar concurrencia y realizar operaciones de red de manera eficiente. Al replicar datos, se busca no solo mover la información, sino también adaptarla al modelo de documentos que ofrece MongoDB, permitiendo así una mejor integración con aplicaciones modernas que requieren escalabilidad y flexibilidad en el manejo de datos.

Este proceso puede incluir la transformación de estructuras de datos, así como la validación y la inserción de documentos en colecciones de MongoDB, todo ello dentro de una arquitectura que facilita la gestión de configuraciones y conexiones entre ambas bases de datos.

Link a la documentación tanto de Go como de MongoDB

- Go: https://go.dev
- MongoDB: https://www.mongodb.com/

Funciones empleadas el código

Conexión a la base de datos de origen:

Una de las funciones clave en el proceso de replicación es la encargada de establecer la conexión con la base de datos de origen. Esta función utiliza las credenciales y parámetros de configuración proporcionados para conectar de manera segura y eficiente. A través de esta conexión, se garantiza que el sistema pueda acceder a los datos que serán replicados, lo que es esencial para el éxito del proceso.

```go
func connectPostgres() (*sql.DB, error) {
    loadEnv()

    // Obtener valores de las variables de entorno
    user := os.Getenv("POSTGRES_USER")
    password := os.Getenv("POSTGRES_PASSWORD")
    dbName := os.Getenv("POSTGRES_DB")
    host := os.Getenv("POSTGRES_HOST")
    port := os.Getenv("POSTGRES_PORT")

    // Construir la cadena de conexión
    connStr := fmt.Sprintf("postgres://%s:%s@%s:%s/%s?sslmode=disable", user, password, host, port, dbName)
    return sql.Open("pgx", connStr)
}
```

Imagen: conexión con postgres (Vscode)

Consulta de migración de datos a MongoDB:

La replicación de una base de datos a MongoDB utilizando Go es un proceso que permite transferir datos desde una base de datos relacional, como PostgreSQL la cual se utilizó para la práctica uno y se tenían cargados todos los datos hacia una base de datos NoSQL, en este caso MongoDB. Este enfoque aprovecha las capacidades de Go para manejar concurrencia y realizar operaciones de red de manera eficiente. Al replicar datos, se busca no solo mover la información, sino también adaptarla al modelo de documentos que ofrece MongoDB, permitiendo así una mejor integración con aplicaciones modernas que requieren escalabilidad y flexibilidad en el manejo de datos.

```go
func migrarDatos(w http.ResponseWriter, r *http.Request, db *sql.DB, collection *mongo.Collection)
    // Consulta principal
    query := "SELECT a.name FROM musicbrainz_old.artist a"
    rows, err := db.Query(query)
    if err != nil {
        http.Error(w, "Error consultando en PostgreSQL", http.StatusInternalServerError)
        return
    }
    defer rows.Close()

    // Iterar sobre los resultados
    for rows.Next() {
        var nombre sql.NullString
        var nombre2 string
        if err := rows.Scan(&nombre); err != nil {
            if err == sql.ErrNoRows {
                log.Printf("Artista %s no encontrado", nombre)
            }
            log.Println("Error al escanear nombre:", err)
            continue
        }

        if nombre.Valid {
            nombre2 = nombre.String
        } else {
            continue
        }
        // Obtener información adicional
        albumes, err := obtenerAlbumesPorArtista(nombre2, w, db)
        if err != nil {
            log.Printf("Error obteniendo álbumes para %s: %v", nombre, err)
            continue
        }

        canciones, err := obtenerCancionesPorArtista(nombre2, w, db)
        if err != nil {
            log.Printf("Error obteniendo canciones para %s: %v", nombre, err)
            continue
        }

        disqueras, err := obtenerDisquerasPorArtista(nombre2, w, db)
        if err != nil {
            log.Printf("Error obteniendo disqueras para %s: %v", nombre, err)
            continue
        }

        artistaInfo, err := obtenerInformacionArtista(nombre2, w, db)
        if err != nil {
            log.Printf("Error obteniendo información del artista %s: %v", nombre, err)
            continue
        }

        // Agregar los atributos adicionales al objeto artistaInfo
        artistaInfo["Albumes"] = albumes
        artistaInfo["Canciones"] = canciones
        artistaInfo["Disqueras"] = disqueras

        // Insertar el documento en MongoDB
        _, err = collection.InsertOne(context.TODO(), artistaInfo)
        if err != nil {
            log.Printf("Error insertando en MongoDB para el artista %s: %v", nombre, err)
            continue
        }
    }
```

Imagen: Migracion de los datos (Vscode)

Función obtenerCancionesPorArtista:

La función obtenerCancionesPorArtista tiene el propósito de extraer las canciones de un artista específico. Similar a la función anterior, recibe el nombre del artista como argumento y ejecuta una consulta SQL que agrupa las canciones por su nombre, obteniendo también la duración de cada canción. Durante la iteración de los resultados, se maneja la conversión de la duración de milisegundos a un formato más legible (minutos y segundos). Los resultados se organizan en un slice de tipo BSON, donde cada canción se convierte en un documento. La función también incluye un manejo de errores para situaciones donde la consulta falle o no se encuentren canciones, asegurando así que se mantenga la integridad del flujo de datos. Al final, retorna el slice de canciones que se incorporará al objeto del artista.

```go
func obtenerCancionesPorArtista(nombre string, w http.ResponseWriter, db *sql.DB) ([]bson.M, 
    // Definir la consulta SQL con el nombre como parámetro
    query := `
        SELECT
            t.name AS nombre,
            MIN(t.length) AS duracion
        FROM
            musicbrainz_old.track t
            JOIN musicbrainz_old.artist a ON t.artist_credit = a.id
        WHERE
            a.name = $1
        GROUP BY
            t.name;`

    // Ejecutar la consulta utilizando la variable 'nombre' como parámetro
    rows, err := db.Query(query, nombre)
    if err != nil {
        http.Error(w, "Error querying postgres", http.StatusInternalServerError)
        return nil, fmt.Errorf("error querying postgres: %v", err)
    }
    defer rows.Close()

    // Crear el slice para almacenar las canciones
    var canciones []bson.M

    for rows.Next() {
        var lengthMillis sql.NullInt64
        var trackName sql.NullString
        var formattedDuration string
        if err := rows.Scan(&trackName, &lengthMillis); err != nil {
            if err == sql.ErrNoRows {
                log.Printf("track %s no encontrado", nombre)
            }
            http.Error(w, "Error al leer fila de PostgreSQL", http.StatusInternalServerError)
            return nil, fmt.Errorf("error al leer fila de PostgreSQL: %v", err)
        }

        if lengthMillis.Valid {
            // Convertir 'lengthMillis' a minutos y segundos
            seconds := lengthMillis.Int64 / 1000 // Convertir milisegundos a segundos
            minutes := seconds / 60              // Calcular los minutos completos
            remainingSeconds := seconds % 60     // Calcular los segundos restantes
            formattedDuration = fmt.Sprintf("%02d:%02d", minutes, remainingSeconds)
        } else {
            formattedDuration = fmt.Sprintf("%02d:%02d", 0, 0)
        }

        // Crear el objeto BSON y hacer append al slice canciones
        cancion := bson.M{
            "Nombre":   trackName.String,
            "Duracion": formattedDuration,
        }
        canciones = append(canciones, cancion)
    }

    // Retornar el slice de canciones
    return canciones, nil
}
```

Imagen: Obtener canciones por artista (Vscode)

Función obtenerDisquerasPorArtista:

La función obtenerDisquerasPorArtista tiene como objetivo recuperar las disqueras asociadas a un artista en particular. Al igual que las funciones anteriores, recibe el nombre del artista como parámetro y ejecuta una consulta SQL que une varias tablas para obtener la información de las disqueras.

Los resultados se almacenan en un slice BSON, donde cada disquera se agrega como un documento. La función también se encarga de manejar errores potenciales durante la consulta o el escaneo de resultados. Una vez que se han recopilado todas las disqueras, se retorna el slice correspondiente, permitiendo que la información se integre en el objeto final del artista durante el proceso de migración.

```go
// Función para obtener disqueras por nombre de artista
func obtenerDisquerasPorArtista(nombre string, w http.ResponseWriter, db *sql.DB) ([]bson.M, error) {
    // Definir la consulta SQL con el nombre como parámetro
    query := `
        SELECT DISTINCT
            lb.name AS disquera
        FROM
            musicbrainz_old.artist a
            LEFT JOIN musicbrainz_old.release r ON r.artist_credit = a.id
            LEFT JOIN musicbrainz_old.language l ON r.language = l.id
            LEFT JOIN musicbrainz_old.release_alias ra ON r.id = ra.release
            LEFT JOIN musicbrainz_old.release_label rl ON r.id = rl.release
            LEFT JOIN musicbrainz_old.label_alias lb ON rl.label = lb.label
        WHERE
            a.name = $1;`

    // Ejecutar la consulta utilizando la variable 'nombre' como parámetro
    rows, err := db.Query(query, nombre)
    if err != nil {
        http.Error(w, "Error querying postgres", http.StatusInternalServerError)
        return nil, fmt.Errorf("error querying postgres: %v", err)
    }
    defer rows.Close()

    // Crear el slice para almacenar las disqueras
    var disqueras []bson.M

    for rows.Next() {
        var disquera label
        if err := rows.Scan(&disquera.Name); err != nil {
            if err == sql.ErrNoRows {
                log.Printf("disquera %s no encontrado", nombre)
            }
            http.Error(w, "Error al leer fila de PostgreSQL", http.StatusInternalServerError)
            return nil, fmt.Errorf("error al leer fila de PostgreSQL: %v", err)
        }

        // Crear el objeto BSON y hacer append al slice disqueras
        disquera2 := bson.M{
            "Nombre": disquera.Name.String,
        }
        disqueras = append(disqueras, disquera2)
    }

    // Retornar el slice de disqueras
    return disqueras, nil
}
```

Imagen: Obtener disqueras por artista (Vscode)

Función obtenerInformacionArtista:

La función obtenerInformacionArtista se especializa en recuperar detalles específicos sobre un artista, como su nombre, año de nacimiento, lugar de nacimiento, tipo, género, estado y biografía. Utiliza una consulta SQL que une la tabla de artistas con la tabla de áreas para obtener la información deseada. Tras ejecutar la consulta, la función escanea los resultados y maneja posibles errores. También incluye lógica para manejar valores nulos en los campos, proporcionando valores predeterminados para el tipo, género y estatus. Finalmente, crea un objeto BSON que encapsula toda esta información sobre el artista, que luego se puede insertar en la colección de MongoDB, completando así la migración de datos.

```go
func obtenerInformacionArtista(nombre string, w http.ResponseWriter, db *sql.DB) (bson.M, error) {
    query := `
    // Ejecutar la consulta utilizando la conexión proporcionada
    row := db.QueryRow(query, nombre)

    var artista artist
    var tipo, genero sql.NullInt64
    var estatus sql.NullBool
    var tipo2, genero2 int64
    var tipo3, genero3, estatus3 string
    var estatus2 bool

    // Escanear los valores de la fila devuelta
    if err := row.Scan(&artista.Name, &artista.Birth, &artista.AreaBirth, &tipo, &genero, &estatus, &artista.Biography); err != nil {
        if err == sql.ErrNoRows {
            log.Printf("Artista %s no encontrado", nombre)
        }
        return nil, fmt.Errorf("error al obtener información del artista: %v", err)
    }

    if tipo.Valid {
        tipo2 = tipo.Int64
        tipo3 = "desconocido"
        if tipo2 == 1 {
            tipo3 = "solista"
        } else if tipo2 == 2 {
            tipo3 = "grupo"
        }
    } else {
        // Asignar un valor por defecto o manejar el caso nulo
        tipo3 = "no aplica"
    }

    if genero.Valid {
        genero2 = genero.Int64
        genero3 = "No aplica"
        if genero2 == 1 {
            genero3 = "Masculino"
        } else if genero2 == 2 {
            genero3 = "Femenino"
        }
    } else {
        // Asignar un valor por defecto o manejar el caso nulo
        genero3 = "No aplica"
    }

    if estatus.Valid {
        estatus2 = estatus.Bool
        estatus3 = "No aplica"
        if estatus2 == false {
            estatus3 = "Activo"
        } else if estatus2 == true {
            estatus3 = "Retirado"
        }
    } else {
        // Asignar un valor por defecto o manejar el caso nulo
        estatus3 = "No aplica"
    }

    // Crear un objeto BSON para el documento del artista
    artista1 := bson.M{
    }

    return artista1, nil
}
```

Imagen: Obtener información del artista (Vscode)

Scripts empleados para la manipulación de datos almacenados en Postgres

Script obtenerAlbumesPorArtista:

Esta consulta selecciona álbumes de un artista específico, utilizando un LEFT JOIN para incluir información sobre el idioma y alias de lanzamientos. Utiliza $1 como parámetro para el nombre del artista, y DISTINCT ON (r.name) asegura que solo se obtenga un álbum por nombre, ordenando por el nombre del álbum y su ID.

```
SELECT DISTINCT ON (r.name)
r.name AS nombre_album,
l.name AS lenguaje,
r.comment AS version
FROM
musicbrainz_old.artist a
LEFT JOIN musicbrainz_old.release r ON r.artist_credit = a.id
LEFT JOIN musicbrainz_old.language l ON r.language = l.id
LEFT JOIN musicbrainz_old.release_alias ra ON r.id = ra.release
WHERE
a.name = $1
ORDER BY
r.name, r.id;
```

Script obtenerCancionesPorArtista:

Obtiene las canciones de un artista específico, utilizando un JOIN para relacionar las canciones con el artista. Agrupa los resultados por nombre de la canción y utiliza MIN(t.length) para obtener la duración mínima (en milisegundos) de cada canción.

```
SELECT
    t.name AS nombre,
    MIN(t.length) AS duracion
FROM
    musicbrainz_old.track t
    JOIN musicbrainz_old.artist a ON t.artist_credit = a.id
WHERE
    a.name = $1
GROUP BY
    t.name;
```

Script obtenerDisquerasPorArtista:

Busca las disqueras asociadas a un artista, usando múltiples LEFT JOIN para conectar tablas

relacionadas con lanzamientos y etiquetas. El resultado es distinto (DISTINCT) para evitar duplicados en los nombres de las disqueras.

```
SELECT DISTINCT
    lb.name AS disquera
FROM
    musicbrainz_old.artist a
    LEFT JOIN musicbrainz_old.release r ON r.artist_credit = a.id
    LEFT JOIN musicbrainz_old.language l ON r.language = l.id
    LEFT JOIN musicbrainz_old.release_alias ra ON r.id = ra.release
    LEFT JOIN musicbrainz_old.release_label rl ON r.id = rl.release
    LEFT JOIN musicbrainz_old.label_alias lb ON rl.label = lb.label
WHERE
    a.name = $1;
```

Script obtenerInformacionArtista:

Recupera información detallada sobre un artista, incluyendo su nombre, año de nacimiento, lugar de nacimiento, tipo, género, estatus y biografía. Utiliza un LEFT JOIN para incluir información sobre la área de nacimiento, con $1 como parámetro para el nombre del artista.

```
SELECT
    a.name AS nombre,
    a.begin_date_year AS anio_nacimimento,
    ar.name AS lugar_nacimiento,
    a.type AS tipo_artista,
    a.gender AS genero,
    a.ended AS estatus,
    a.comment AS biografia
FROM
    musicbrainz_old.artist a
    LEFT JOIN musicbrainz_old.area ar ON a.area = ar.id
WHERE
    a.name = $1;
```