Proyecto de Fin de Curso: Herramienta de migración desde una base en MySQL a una base de MongoDB

Julieta Dubra, *4972820-7*
Ramiro Bentancor, *5126656-8*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay

I. Introducción

Las bases de datos documentales son un subtipo específico de base de datos no relacional que almacena bases de datos como documentos estructurados (como XML o JSON). Durante el curso, se utilizó MongoDB como sistema de bases documentales, y se vieron posibles formas de pasar diseños de bases relacionales a diseños no relacionales. Se buscó, entonces, crear una herramienta capaz de automatizar este proceso y de migrar los datos entre una base de datos relacional de MySQL y una documental en MongoDB. Para eso, se implementó un algoritmo que consta de 2 partes: la creación del nuevo esquema de MongoDB y luego la migración de los datos entre ambas bases.

II. Trabajos relacionados

El trabajo se basa fuertemente en el paper "Automatic Mapping of MySQL Databases to NoSQL MongoDB" [Lia], que describe cómo implementar un algoritmo original que permite crear el esquema de la base documental usando las tablas del INFORMATION_SCHEMA de MySQL. El mismo define cómo se representará en el esquema documental, cada tabla de la base de datos de MySQL, en función de cuantas tablas referencia (cantidad de claves foráneas que posee) y por cuantas tablas es referenciada. Define entonces un criterio de clasificación, como se puede ver en el cuadro 1.

Tabla en MySQL	**Representación en MongoDB**
Tabla no es referenciada por otras tablas	Colección
Tabla sin claves foráneas, referenciada por otra tabla	Colección
Tabla con una clave foránea, referenciada por otra tabla	Colección, utiliza el método de linking
Tabla con una clave foránea, no es referenciada por otras tablas	Utiliza el método de embeded.
Tabla con dos claves foráneas, no es referenciada por otras tablas	Two way embedding model
Tabla con tres o más claves foráneas	Utiliza el método de linking

Cuadro I: Clasificación de tablas por el algoritmo del paper

Se definen en esta tabla tres métodos posibles para representar las relaciones:

- Linking: se relacionan los documentos, se mantiene la clave foránea como un campo del documento. Este método evita la redundancia de datos, pero vuelve algunas consultas más costosas.
- Embedding: se incluye el objeto completo dentro de otro. Este método hace que las consultas sean menos costosas, pero las modificaciones y las inserciones se ven afectadas.
- Two way embedding: este método es una variante del embedding, en la que se realiza el embedding en las dos colecciones relacionadas.

El algoritmo apunta a definir un método estándar que balancea las distintas técnicas para conseguir una solución lo más eficiente posible. Sin embargo, al realizar la solución presentada, se consideró que una buena solución se adapta a cada caso. Por lo tanto, se buscó agregar cierta flexibilidad al programa, permitiendo al usuario elegir ciertas opciones que serán descritas más adelante. Se incluye además el posible uso de dos patrones descritos en " Building with Patterns: A Summary" [Mona], para poder aplicar estas optimizaciones en los casos que corresponda.

III. Description de la herramienta

Se implementó una herramienta usando el entorno de javascript Node js. Se consideró adecuada ya que la JavaScript Object Notation (JSON) es la notación usada para manejar objetos en javascript, y una variante de JSON es usada por MongoDB. Se

consideró entonces que con Node Js se facilita el cambio entre objetos de la base relacional a objetos de la base documental. La herramienta es capaz de realizar migraciones, y además de medir el tiempo de ciertas consultas de Mongo.

La herramienta de migración fue implementada en 3 partes. La primera implementa una interfaz de consola, que le pide al usuario los datos de conexión a la base, y las opciones que quiere usar. La segunda, crea la conexión a MySQL y obtiene los datos para definir el nuevo esquema de la base de datos. Finalmente, la última parte realiza la migración de los datos entre ambas bases. Se verán entonces en detalle cada una de las partes.

III-A. Link al repositorio conteniendo el trabajo

Se incluye el link al repositorio que contiene el código de la herramienta implementada:

`https://gitlab.fing.edu.uy/ramiro.bentancor/bdnr-node`

Una vez clonado el repositorio, se debe correr el siguiente comando en la terminal para instalar los paquetes:

```
npm i
```

Luego, para correr el programa, se usa el comando:

```
npm run start
```

III-B. Interfaz de consola

La interfaz de consola es muy simple y fue realizada utilizando la librería prompts [NPM] de Node js. La interfaz le pide primero al usuario que inserte los datos necesarios para establecer la conexión con MySQL y luego los de MongoDB. Estos datos son:

- El nombre de la base de datos a migrar. La base creada en MongoDB tendrá este mismo nombre.
- El nombre de usuario de MySQL.
- La contraseña para el usuario de MySQL.
- El host en el que se encuentra corriendo la base de datos. El valor por defecto es 'localhost'.
- La URI usada para conectarse a Mongo. El valor por defecto es 'mongodb://localhost:27017', valor usado para conectarse a una base local por defecto.

Esto le da un cierto nivel de flexibilidad inicial al usuario. No solo tiene la capacidad de poder migrar la base que el quiera desde un mismo host, pero además puede decidir si las conexiones a Mongo y MySQL son locales, o a hosts exteriores. Por lo tanto, se podría por ejemplo migrar una base de MySQL local a un servidor externo que contenga una base de Mongo. Sin embargo, la verdadera flexibilidad es implementada luego, cuando se le pide al usuario que ingrese que quiere hacer respecto a 4 opciones. Estas son:

- Usar embedding simple en vez de two way embedding. Si se selecciona esta opción, por cada relación que es clasificada como two way embedding, el programa le pregunta al usuario que quiere hacer. Puede decidir qué tabla generará una subcollection dentro de la otra, o decidir mantener el embedding doble. Esto maximiza la flexibilidad de la opción porque quizás es mejor tener algunos casos con embedding doble, pero no todos. Esto permite definir, relación a relación, que se hará para cada una.
- Usar linking en vez de embedding simple. Si se elige esta opción, para todas las tablas clasificadas como embedding simple (tabla con una clave foránea, que no es referenciada por otras tablas), se da la opción de en vez usar linking. De la misma forma que con la opción anterior, se pregunta por cada tabla de ese tipo si se quiere hacer embedding o linking, pudiendo decidir caso por caso.
- Usar el Extended Reference Pattern [Monb]. Dentro de la aplicación, este patrón puede ser usado en los casos donde se usa two way embedding. En estos casos, se crea una colección independiente para cada tabla, y se incluye cada una dentro de la otra. Si se activa la opción 1, igual tenemos dos tablas independientes con todos los datos, solo se puede cambiar como hacer las inclusiones. Por lo tanto, en estos casos, tiene sentido que el usuario pueda elegir qué datos quiere incluir en las subcolecciones generadas, ya que las mismas contienen información redundante. Usando esta opción, se le pregunta al usuario para todas las subcolecciones que va a generar, que columnas de la misma quiere incluir, pudiendo elegir así cuales son los datos realmente necesarios.
- Usar el Schema Versioning Pattern [Monc]. Este patrón se puede usar en caso que se crea que vayan a haber cambios en el esquema a futuro. El mismo le agrega un campo, 'schema_version' con valor 1, a cada colección de mongo que se crea. Este patrón permite que existan, simultáneamente, versiones previas y nuevas del documento.

Estas opciones modifican ligeramente el algoritmo descrito previamente, pero la clasificación que realiza es la misma. Los cambios se implementan cuando se crea el esquema para Mongo, o cuando se están migrando los datos en caso de el Schema Versioning Pattern.

III-C. Creación del nuevo esquema

Para crear el nuevo esquema, primero se consigue la información sobre las tablas de la base de datos, de forma a poder clasificarlas según el algoritmo. Para eso se realizan consultas al INFORMATION_SCHEMA de MySQL. Primero, se consiguen todas las tablas de la base con la siguiente consulta:

```
SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = database AND TABLE_TYPE = 'BASE TABLE';
```

Luego para cada tabla, se consiguen las claves foráneas que tienen, y por qué tablas son referenciadas, llegando a una colección de objetos de la siguiente forma:

```
{
 name,
 fields,
 foreignKeys,
 externalForeignKeys,
 type,
}
```

Donde el tipo es la clasificación de la tabla según el algoritmo, sus valores posibles siendo 1) colección 2) linking 3) embed 4) two way embed. Una vez conseguida esta información, se procesan los datos para definir el esquema de Mongo que se va a crear. Para esto se crea una nueva colección llamada 'schema', que contiene objetos de la siguiente forma:

```
{
 name,
 fields,
 subcollections,
}
```

Donde para cada colección que se tendrá en Mongo, se tendrá un objeto en 'schema', que contiene su nombre y los campos que tiene. Para los tipos 1 y 2, crear un nuevo objeto de este tipo con subcolecciones inicialmente vacías es todo lo que hace falta hacer. Para los tipos 3 y 4 se empiezan a crear objetos del siguiente tipo, que son incluidos en los correspondientes arrays de 'subcollections':

```
{
 name,
 foreignKey,
 localKey,
 type,
 fieldString,
 getData,
}
```

Donde se generan los distintos datos necesarios para crear la subcoleccion. Las subcolecciones se crean según las opciones elegidas: embedding doble pueden ser dos, o una sola, por ejemplo. Además, el campo 'fieldString' contiene un string que puede ser incluido en una consulta de MySQL con las columnas que deben estar en la subcolección (elegidas con la opción del Extended Reference Pattern). Cabe aclarar que las subcolecciones contienen un campo tipo, que indica si se debe aplicar el método de migración creado para el embedding simple o doble.

Una vez que se consiguen estos datos, se llama a la función "createMongoCollections", que crea una versión inicial del schema en la base de mongo, sobre la cual se puede luego agregar datos.

III-D. Migración de datos

Esta última parte del programa realiza, efectivamente, la migración de datos entre las dos bases. Para esto itera cada elemento de la colección pre procesada 'schema' para ir migrando los datos de la nueva colección. De forma de no exceder la memoria disponible para el programa, primero se cuentan cuántas filas hay en cada tabla, y se van ingresando los datos en grupos de hasta quinientos filas. Si una colección va a tener subcolecciones, luego de la inserción de cada fila, la misma es inmediatamente modificada llamando a funciones que insertan todos los valores de las subcolecciones que deben incluir. De la misma forma que con las tablas, las tablas usadas para las subcolecciones son migradas de a quinientos objetos por vez.

Como se mencionó previamente, si una tabla tiene subcolecciones se puede llamar uno de dos métodos posibles: uno para el embedding y otro para el two way embedding. Esto se debe principalmente a que en el two way embedding hay que ir a conseguir los datos a una tabla representante de una relación, para luego ir a las tablas correspondientes a buscar los datos. En embedding, uno puede ir directamente a la tabla a buscar los datos.

Una vez terminada la migración, el programa notifica al usuario y termina su ejecución. La nueva base en MongoDB queda disponible.

III-E. Consultas en MongoDB

Para realizar algunas pruebas, la herramienta es capaz de medir el tiempo que toma la ejecución de consultas que utilizan la agregación pipeline. Para eso, el usuario debe ingresar el nombre de la base, y el nombre de la colección en que se realizará la consulta. El pipeline se obtiene del archivo 'query.js' que se encuentra en la carpeta 'src'. La idea es que el usuario pueda exportar el pipeline desde MongoDB Compass en formato para Node Js, pegar el resultado en este archivo y ver cuánto tiempo toma en correr la consulta.

IV. Experimentación

Se hicieron una serie de tests para probar el funcionamiento del programa. Se pueden dividir en 3 grandes categorías. Primero, se realizaron distintas migraciones con distintas bases, probando que el programa funcione correctamente para todas las opciones posibles. Luego, se realizó un test con hosts distintos a localhost para ver que el programa pueda establecer conexiones a otras plataformas. Finalmente, se realizaron consultas sobre las bases resultantes para comparar cómo las distintas opciones pueden afectar a la performancia de ciertas consultas.

IV-A. Test de migraciones

Para hacer estos tests, se utilizaron 3 bases de datos distintas. En primer lugar, una base muy simple llamada country_cities, que se creó específicamente para realizar pruebas con pocos datos, y para testear el embedding simple. Luego se usó la base sakila, para utilizar un ejemplo con más datos y que incluya two way embedding. Finalmente se usó una base pre cargada, employees, que contiene una cantidad bastante más importante de datos. Esta base se usó para mostrar cómo los distintos esquemas definidos pueden agilizar o enlentecer las migraciones, ya que pueden definir la cantidad de redundancia en los datos.

Figura 1: Esquema de la base country_cities.

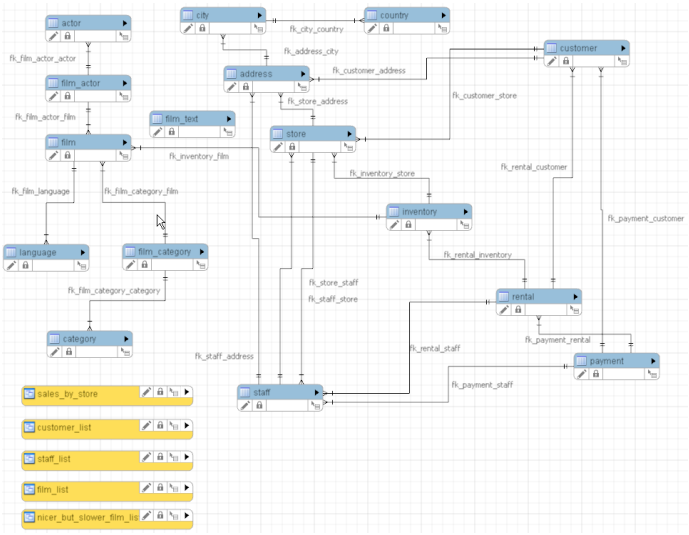

Figura 2: Esquema de la base sakila.

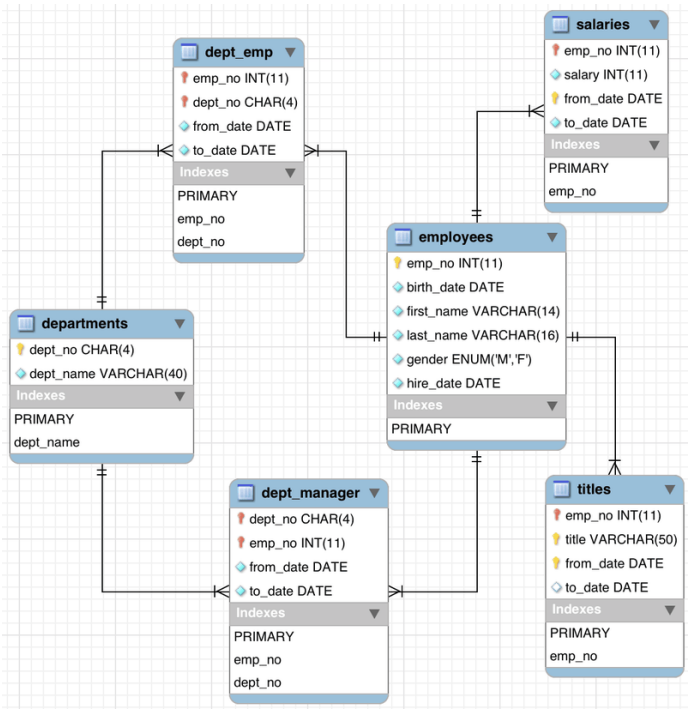

Figura 3: Esquema de la base employees.

Se siguió el siguiente plan de pruebas, donde la opción 1 es usar simple en vez de two way embedding y la opción 2 es usar linking en vez de simple embedding.

	Base de datos	Opción 1	Opción 2	Extended reference	Schema versioning
1	country_cities	no	no	no	no
2	country_cities	no	yes	no	no
3	country_cities	no	no	no	yes
4	sakila	no	no	no	no
5	sakila	yes	no	no	no
6	sakila	no	no	yes	no
7	sakila	no	no	no	yes

Cuadro II: Plan de pruebas

Se indica para cada caso las opciones que se usaron para realizar las pruebas. Las primeras 3 pruebas se realizaron sobre country_cities, haciendo primero todo con embedding, luego todo con linking, y luego testeando que funcione el schema versioning. Estas pruebas fueron simples, las tablas teniendo ambas muy pocas filas. Fueron unas pruebas iniciales, realizadas durante el desarrollo para ver que estuviera funcionando bien desde un principio.

Se realizaron una serie de pruebas más avanzadas con la base sakila, probando las otras opciones y además probando más de una opción simultáneamente. Estos tests fueron más robustos, y se hicieron más cerca del final del desarrollo. Todas las migraciones planteadas fueron exitosas, y en algunos casos se realizaron los tests múltiples veces, eligiendo distintas tablas para embeber o distintos atributos para incluir en las subcolecciones.

Finalmente, se van a ver en más detalle los últimos dos test realizados. Estos se hicieron sobre la base de datos employees [Ora]. Como se puede ver en la misma, cada empleado pertenece a un departamento, y cada departamento tiene un conjunto muy pequeño de directores. Ambas relaciones, empleado-departamento, y departamento-director, son clasificadas como two way embedding por el algoritmo. Sin embargo, si se aplica realmente el two way embedding (como en la prueba 9), la migración es bastante más lenta ya que se deben ingresar todos los empleados de nuevo en algún departamento para la relación empleado-departamento, y además se debe chequear para cada empleado si es director de algún departamento. En la prueba 10 entonces se eligió usar embedding simple en vez de doble, incluyendo la tabla departamento en empleados, y los directores dentro de los departamentos. Esto agiliza la migración, ya que cada empleado no director es ingresado una vez y los directores (que son muy pocos) son los únicos ingresados dos veces. Además, se usó el Extended Reference Pattern para limitar los datos

de los empleados que se incluyen a su id y su nombre, incluyendo una cantidad mucho menor de datos. Esta migración fue más rápida, y generó un modelo de datos mucho más eficiente del que se habría generado por el algoritmo ‘default’.

IV-B. Test de plataformas

Una de las funcionalidades que provee la aplicación, es la posibilidad de conectarse de manera remota tanto a la base de MySQL como a la de MongoDB. Como ya se mencionó, una de las opciones que permite configurar la interfaz de consola, son los hosts de las conexiones a ambas bases. Para la conexión remota de MySQL se utilizaron dos máquinas (computadoras) distintas con el fin de simular un servidor o host externo. En este caso es necesario habilitar inicios de sesión remotos, en el servidor MySQL que provee la conexión.
Por otro lado, la conexión remota de MongoDB fue testeada utilizando un clúster ubicado en MongoDB Atlas. De parte del clúster, es necesario que el usuario con el que se vaya a acceder tenga permisos de administrador. Para efectuar esta conexión, en lugar de usar el valor por defecto, se puede copiar el URI de la conexión proporcionado por Atlas.

IV-C. Test de performance en consultas

Se realizaron finalmente dos consultas en dos migraciones distintas de la tabla ‘employees’, para ver cómo varía la velocidad de las consultas. La primera base, es la generada por la prueba 10. La segunda es una base muy similar a ésta, salvo que usa linking entre las tablas de salarios y la de empleados, y entre la de títulos y empleados.

La primera consulta que se realizó, en ambas bases, fue obtener todos los salarios que cobró un empleado dado. Para la primera base, esa consulta quedo como:

```
[
  {
    '$match': {
      'emp_no': 10001
    }
  }, {
    '$project': {
      '_id': false,
      'salaries': true
    }
  }
]
```

Y para la segunda quedó como:

```
[
  {
    '$match': {
      'emp_no': 10001
    }
  }
]
```

Al correr las dos consultas, se noto que la primera fue notablemente más rápida que la segunda: esto pasa ya que en la primera solo debe devolver la subcolección de un campo, mientras en la segunda debe recorrer todos los salarios y devolver los que corresponden al empleado indicado. Por lo tanto, para esta consulta el embedding es mejor.

Luego, se realizó una consulta que busca devolver todos los salarios emitidos después de cierta fecha. Para la primera base, esa consulta quedo como:

```
[
  {
    '$unwind': '$salaries'
  }, {
    '$match': {
      'salaries.from_date': {
        '$gte': '1987-06-26'
      }
    }
```

```
  }, {
    '$project': {
      '_id': false,
      'salaries': true
    }
  }
]
```

Y para la segunda quedó como:

```
[
  {
    '$match': {
      from_date: {
        '$gte': '1987-06-26',
      }
    }
  },
]
```

En este caso, la consulta tomó un poco menos de tiempo en la segunda base, ya que en ambos casos hay que recorrer aproximadamente la misma cantidad de registros (todos los salarios), pero en la segunda base no hace falta hacer el unwind. Los tiempos son sin embargo bastante más cercanos que en la primera consulta. Se considera, sin embargo, que esta consulta podría ser fuertemente optimizada para la segunda base si se incluyera un índice como se puede tener en MySQL, mejorando así la velocidad respecto a la del servidor 1.

V. Conclusiones y trabajo futuro

Se puede decir, para terminar, que se implementró una herramienta capaz de migrar bases desde MySQL a MongoDB que sigue de bastante cerca el algoritmo descrito en el paper. Sin embargo, la implementación realizada presenta una ventaja principal sobre la implementación del paper que es su flexibilidad. Como se vio en los ejemplos, un buen esquema puede no solo agilizar las consultas pero además las migraciones.

Se considera que la herramienta tiene mucho potencial de crecimiento. Se podría expandir para que sea capaz de migrar otros aspectos de las bases relacionales, como por ejemplo índices. Además, se podría adaptar para que sea capaz de migrar bases relacionales desde otros sistemas de manejo de datos, como PostgreSQL. Además, se podría hacer que el programa sea todavía más flexible, pudiendo agregar más opciones como cambiar el embedding doble por linking, o el linking por embedding simple, etc. Finalmente, se considera que la aplicación podría tener una interfaz más amigable para el usuario, o ser transformada a forma de API que se publique como servicio de forma a poder ser usada sin tener que correr el código localmente.

REFERENCIAS

[Lia] Dumitru Dan Burdescu Liana Stanescu Marius Brezovan. *Automatic Mapping of MySQL Databases to NoSQL MongoDB*. URL: https://annals-csis.org/Volume_8/pliks/pliks/45.pdf.

[Mona] MongoDB. *Building with Patterns: A Summary*. URL: https://www.mongodb.com/blog/post/building-with-patterns-a-summary. (accedido: 09.05.2022).

[Monb] MongoDB. *Building with Patterns: The Extended Reference Pattern*. URL: https://www.mongodb.com/blog/post/building-with-patterns-the-extended-reference-pattern.

[Monc] MongoDB. *Building with Patterns: The Schema Versioning Pattern*. URL: https://www.mongodb.com/blog/post/building-with-patterns-the-schema-versioning-pattern.

[NPM] NPM. *Librería Prompts de Node js*. URL: https://www.npmjs.com/package/prompts.

[Ora] Oracle. *Employees Sample Database*. URL: https://dev.mysql.com/doc/employee/en/employees-introduction.html.