Implementación de algoritmo para migración de base de datos en MySQL, a MongoDB

Nicolás Cardozo *Instituto de Computación*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
adrian.nicolas.cardozo@fing.edu.uy
Oscar Saldías *Instituto de Computación*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
oscar.saldias@fing.edu.uy

Resumen

Este proyecto consiste en el desarrollo de una solución de software que permita realizar la migración de una base de datos relacional, específicamente MySQL, a una base no relacional como es MongoDB, usando como guía para su implementación un trabajo de investigación externo.

I. Introducción

El modelo relacional ha sido la solución de facto por mucho tiempo a la hora de cubrir las necesidades de almacenamiento de información. Según Bhat and Jadhav (2010) es la piedra fundacional de muchas pilas de aplicaciones usadas hoy en día, ofrece a los usuarios un equilibrio entre simplicidad, robustez, flexibilidad, rendimiento, escalabilidad y compatibilidad. Sin embargo las tecnologías emergentes centradas en la nube han planteado varios retos en los *Manejadores de Bases de Datos Relacionales* (*MBDR*). Estos no se ajustan a todos los grandes requerimientos de esta nueva generación de aplicaciones, tales como manejar grandes volúmenes de datos no estructurados, o proveer escalabilidad elástica. Debido a esto, como resultado surgen las bases de datos orientadas a documento, u orientadas a recurso, que por naturaleza son no relacionales.

Este trabajo toma como punto de partida las necesidades de la nueva generación de aplicaciones, centrándose en el proceso de mudanza del modelo relacional al no relacional. Más específicamente, se implementará una solución que a partir de estructuras y datos existentes en una base de datos (*BD*) en MySQL[1], permita migrar a sus respectivas en MongoDB[2]. El proceso de migración se basará en el *framework* descrito en Stanescu et al..

Stanescu et al. utiliza un diseño de base de datos de origen, para mostrar aplicación del algoritmo (sec. II), que puede no ser representativa de soluciones productivas. Como parte de los objetivos propuestos en este trabajo, luego de realizada la implementación (sec. III), será migrar la base de datos de prueba Sakila[3] (sec. IV), para luego analizar objetivamente los resultados logrados, problemas detectados, posibles mejoras a realizar y trabajo futuro (sec. V).

II. Trabajos relacionados

El abordaje e implementación de este trabajo se basó fundamentalmente en el algoritmo propuesto por Stanescu et al., el cual describe el proceso de mapeo automático de una base de datos en MySQL a MongoB, un *MBDR* no relacional orientado a documento. El proceso descripto en dicho artículo usa los metadatos almacenados en las tablas del sistema de MySQL, tomando en consideración los conceptos del Modelo Entidad-Relación (MER): tipo de entidad representado por una relación en el modelo relacional, relaciones del tipo 1:1 y 1:M, representados con claves foráneas y del tipo N:M modelado a través de tablas unión (o en inglés *join tables*), que contiene la clave primaria de cada una de las tablas de origen. Cada una representando la clave foránea (CF) y dos relaciones de tipo 1:M entre las tablas de origen y la de unión. Este tipo de estructuras representan relaciones múltiples entre distintas tablas, y cada una podría tener diferentes esquemas de registros.

Cabe mencionar, que el tamaño de la base de datos utilizada en Stanescu et al. constó de 10 a 15 tablas con muchas relaciones y 100 registros por tabla.

III. Trabajo realizado

El enfoque del trabajo se basó en las etapas típicas del proceso de desarrollo de aplicaciones: el análisis de la solución, la investigación de las tecnologías existentes y elección de la o las más adecuadas para su implementación, luego el desarrollo de la solución propiamente dicha, y finalmente experimentación, donde se incluyen las pruebas y ajustes necesarios.

[1] https://www.mysql.com/
[2] https://www.mongodb.com/
[3] https://dev.mysql.com/doc/sakila/en/

III-A. Análisis

Lo primero que se identificó, durante el proceso de análisis de requerimientos y división del problema a resolver, fue la existencia de tres etapas principales por las que debería pasar el algoritmo de migración:

- *Extracción de los metadatos de origen y clasificación:* Obtención de la información correspondiente a las estructuras estáticas (tablas, columnas, filas, tipos de datos, relaciones, etc) de la BD de origen y clasificación utilizando el algoritmo de mapeo que se detalla en la sección III-B.
- *Migración de estructuras:* Creación de la BD de destino y sus estructuras, según clasificación obtenida en el paso anterior.
- *Migración de datos:* Obtención de datos de la BD de origen, conversión de formato según cada tipo (texto, numérico, fecha, tipo de datos lógico, etc) a su correspondiente en MongoDB y almacenamiento en la BD de destino.

Para la primera etapa fue necesario tomar como punto de partida los elementos estructurales de cada uno de los modelos: el relacional y el documental, y qué correspondencia o mapeo existe entre cada uno de ellos. Para ello se utilizó el cuadro provisto por Stanescu et al. (cuadro I), que presenta los principios generales del mapeo relacional a MongoDB, considerando los elementos principales del modelo Entidad-Relación. Como se puede observar, el concepto de base de datos en ambos modelos es el mismo, las tablas que integran una base de datos en el modelo relacional se mapean a colecciones, que también conforman las bases de datos en MongoDB, las filas de una tabla se mapean a documentos y las columnas a campos dentro del documento.

Debido a que en MongoDB no existe el concepto de esquema, las colecciones no tienen un formato o estructura fija que deban cumplir los documentos que aloje. Por esta razón no existe necesidad de realizar una migración de estructuras previo a la migración de datos. Esto resulta en la unión de las dos últimas etapas descritas anteriormente, quedando de la siguiente manera:

- *Extracción de los metadatos de origen y clasificación:* Obtención de la información correspondiente a las estructuras estáticas (tablas, columnas, filas, tipos de datos, relaciones, etc) de la BD de origen y clasificación utilizando el algoritmo de mapeo que se detalla en la sección III-B.
- *Generación de estructuras y migración de datos:* Obtención de datos de la BD de origen, generación de documentos y colecciones según clasificación generada en el paso anterior, conversión de formato según cada tipo (texto, numérico, fecha, tipo de datos lógico, etc) a su correspondiente en MongoDB, y finalmente el almacenamiento en la BD de destino.

III-B. Algoritmo de mapeo

El algoritmo de mapeo automático utilizado es el propuesto en Stanescu et al., que toma como entrada la cantidad de relaciones entrantes y salientes entre las tablas de la BD de origen. Es decir, dada una tabla a migrar, el algoritmo considera la cantidad de referencias externas a dicha tabla (tablas externas con claves foráneas a la tabla), más la cantidad de tablas externas referenciadas por esta (claves foráneas a otras tablas), para decidir cómo debe mapearse a MongoDB, si es a través de una colección, embebiendo documentos, etc.

Antes de describir el algoritmo, se introducen algunos patrones que se manejan en el mismo (entre paréntesis se muestra el nombre con el que aparece en Stanescu et al.):

- *Modelo de enlace (Linking Model):* Este patrón en MongoDB corresponde a la Referencia Manual (Database References — Manual References - [MongoDB1]), es la inclusión del valor del campo *_id* de un documento, en otro documento que lo referencia.
- *Embebido en un sentido (One Way Embedding):* En este caso los registros de dos tablas distintas que tienen relación funcional entre ellas, son migrados a un único documento en un formato de anidación. Es decir, el registro de la tabla que tiene la relación funcional de origen se mapea a un documento en MongoDB y el registro de la tabla que es destino de la relación funcional, es agregado en el documento inicial como un nuevo campo anidado. Por más información Model One-to-One Relationships with Embedded Documents - [MongoDB2] y Model One-to-Many Relationships with Embedded Documents - [MongoDB3].

Cuadro I: Mapeo entre entidades de Manejador de Base de Datos Relacional (MBDR) y MongoDB (Stanescu et al.).

MBDR	***MongoDB***
Base de Datos	Base de Datos
Tabla	Colección
Tupla/Registro	Documento
Columna	Campo
Tabla unión	Documentos embebidos
Clave primaria	Clave primaria (campo provisto por defecto por MongoDB: _id)

- *Embebido en dos sentidos (Two Way Embedding):* Es un caso particular del "embebido en un sentido", se aplica en la migración de tablas que representan relaciones del tipo N:M, donde ambos lados de la relación funcional utilizan el patrón mencionado anteriormente.

A continuación se detalla cada paso en el orden propuesto en el algoritmo original, y su correspondiente representación como salida:

Algoritmo 1 Pseudocódigo del algoritmo de mapeo.

```
Mientras exista tabla T a migrar:
       Si T no es referenciada por otras tablas:
              Migrar como Colección
       Fin Si
       Si T es referenciada por una tabla y no tiene CFs:
              Migrar como Colección
       Fin Si
       Si T es referenciada por una tabla y tiene una CF:
              Migrar como Colección + Modelo de enlace (Linking Model)
       Fin Si
       Si T no es referenciada por otras tablas y tiene una CF:
              Migrar usando Embebido en un sentido (One Way Embedding)
       Fin Si
       Si T no es referenciada por otras tablas y tiene dos CFs:
              Migrar usando Embebido en dos sentidos (Two Way Embedding)
       Fin Si
       Si T tiene tres o más CFs:
              Migrar usando Modelo de enlace (Linking Model)
       Fin Si
Fin Mientras
```

En el cuadro II se puede ver una representación tabular del algoritmo descrito y debajo se detallan los casos que el algoritmo original no cubre. Para las combinaciones que no estaban cubiertas se decidió mapearlo a la representación más genérica, esto es, la tabla se mapea a una nueva colección y se utiliza el modelo de enlace para las tablas que la referencian.

III-C. Tecnologías utilizadas

Para la implementación de la solución que finalmente realice el trabajo de migración se seleccionó Pentaho Data Integration[4], también llamado Kettle, es una tecnología basada en Java, formada por un conjunto de herramientas y aplicaciones que proveen capacidades de extracción, transformación y carga de datos (ETL), con soporte de múltiples tipos de orígenes de datos. Estas características la hacen ideal para aplicaciones de este estilo.

En particular durante el proceso de diseño se utilizó Spoon, que es una herramienta gráfica similar a un ambiente de desarrollo integrado (por sus siglas en inglés IDE) y que forma parte de Kettle.

III-D. Implementación

Como se mencionó anteriormente (sec. III-C), se utilizó Pentaho Data Integration como plataforma de desarrollo y ejecución, ya que es una aplicación Java y por lo tanto permite portabilidad entre distintos sistemas operativos. Además, esta tecnología ajusta a las necesidades del caso de uso, dado que naturalmente está pensada para extraer, transformar y guardar datos entre distintas fuentes. Para el caso concreto de este trabajo utilizamos conectores MySQL, usados en la extracción, y componentes Java para guardar los datos en MongoDB.

[4]https://help.hitachivantara.com/Documentation/Pentaho/7.1/0D0/Pentaho_Data_Integration

Cuadro II: Representación del algoritmo de mapeo propuesto en Stanescu et al. y debajo la extensión a casos no contemplados.

Orden de evaluación	***Cantidad referencias desde tablas externas***	***Cantidad claves foráneas***	***Mapeo***
1	0	Cualquiera	Colección
2	1	0	Colección
3	1	1	Colección + Modelo de enlace
4	0	1	Embebido en un sentido
5	0	2	Embebido en dos sentidos
6	Cualquiera	≥ 3	Colección + Modelo de enlace
Casos no contemplados en algoritmo original			
7	> 1	0	Colección + Modelo de enlace
8	> 1	1	Colección + Modelo de enlace
9	> 0	2	Colección + Modelo de enlace

Allí se desarrollaron dos módulos de transformación (Clasificación y Migración) y un módulo principal que realiza el control y la ejecución de la migración.

A continuación se detalla un pseudocódigo de los módulos implementados

Algoritmo 2 Pseudocódigo del Módulo Principal.

```
Modulo Principal:
       Configuracion de variables iniciales a partir del archivo "mysql2mongodb.properties"
       Ejecutar "Modulo_Clasificacion"

       Mientras el archivo "Clasificacion.csv" contenga tablas:
              Ejecutar "Modulo_Migracion"
       Actualizar archivos temporales: "Clasificacion.csv" y "Tablas_migradas.csv"
   Fin Mientras
Fin Modulo Principal
```

Algoritmo 3 Pseudocódigo del Módulo Clasificación.

```
Módulo Clasificación:
       Obtener tablas disponibles en BD relacional y sus referencias

       Para cada tabla T obtenida:
       Clasificar T según algoritmo detallado en cuadro II, pudiendo obtener los siguientes valores:
          COLLXX: Corresponde migrar como colección, donde XX referencia al punto del artículo (por ejemplo, 22 (2.2), 23 (2.3), etc).
             El valor 29 corresponde a las tablas que no caen en ninguno de los puntos contemplados
          LINKMO: Corresponde migrar usando patrón Linking Model
          W1EMB: Corresponde migrar usando patrón One Way Embedding
          W2EMB: Corresponde al método Two Way Embedding.
       Fin Para

       Guardar clasificación en archivo "Clasificacion.csv"
Fin Modulo Clasificación
```

Algoritmo 4 Pseudocódigo del Módulo Migración.

```
Módulo Migración:
   Obtener tablas disponibles en archivo "Clasificacion.csv"
   Para cada tabla T obtenida:
       Verificar si la tabla cumple las condiciones para ser migrada: todas sus "hijas" (tablas referenciadas con CFs) fueron migradas,
          es decir, está listada en el archivo "Tablas_migradas.csv".

       Si T cumple con las condiciones:
              Marcar el campo Migrar = Y
       Sino
              Marcar el campo Migrar = N
          Fin Si
       Fin Para

       Guardar las tablas marcadas Migrar = N en archivo "Clasificacion.csv"
       Guardar las tablas marcadas Migrar = Y en archivo "Tablas_migradas.csv"

       Para cada tabla T con valor Migrar = Y
              Enviar los datos de T al flujo de migración que corresponde, según clasificación
       Fin Para
Fin Módulo Migración
```

Seguido, se detallan los flujos de migración usados en el proceso de migración (algoritmo 4):

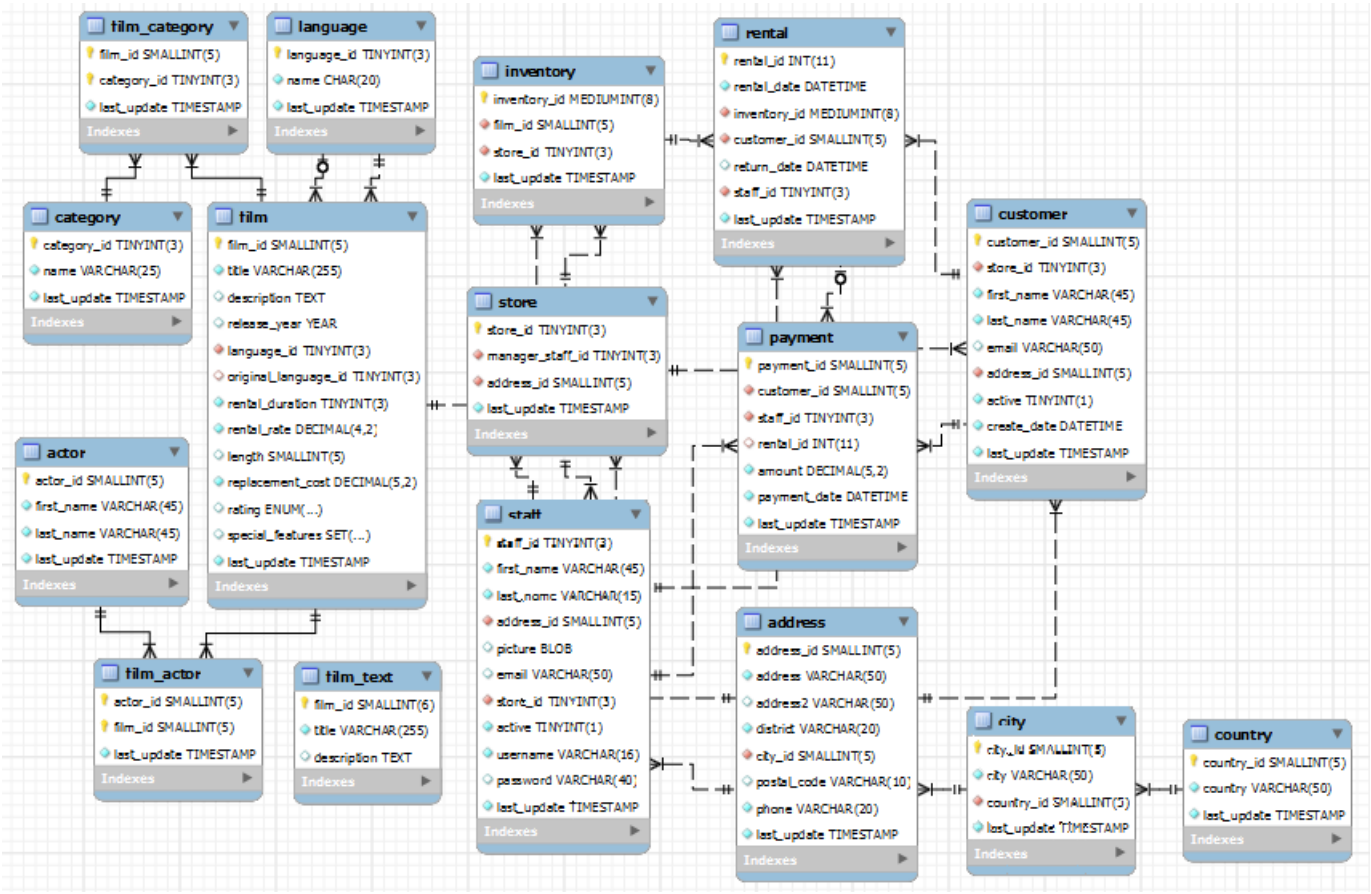
- *COLL:* Toma como entrada las tablas marcadas con COLLXX. Para cada una de estas genera una línea por cada fila de la tabla, con el formato campo,Tipo dato,valor;campo,Tipo dato,valor;.... Luego, estas líneas son transformadas en JSON y cargadas en la colección correspondiente en MongoDB.
- *W1EMB:* Se genera una línea por cada fila de la tabla con el formato campo,Tipo dato,valor;... y en una columna separada se guarda el valor de la CF. Cada fila se transforma en JSON, se obtiene el documento correspondiente al valor de la CF, y se realiza una carga del JSON generado. Esto hace que se embeba el documento generado en el documento correspondiente a la CF.
- *W2EMB:* Similar al punto anterior. La diferencia está en cómo se manejan dos CF, se obtienen los dos documentos de cada una de las colecciones y se realiza una carga cruzada, es decir, el documento 1 de la colección 1 se agrega en el documento 2 de la colección 2, y viceversa. Además, se agregan en ambos documentos todos los campos existentes en la tabla que se está migrando.
- *LINKMO:* Para cada fila de la tabla, tomando como partida las CFs, se buscan los documentos correspondientes en la BD de MongoDB, y se obtienen los valores $_id, con los cuales se reemplaza el valor original.

Cabe mencionar que para en las fases de clasificación y migración fue necesario acceder a los metadatos de las tablas (restricciones referenciales, campos, tipos, etc) usando las tablas del Manejador de Base de Datos Relacional de MySQL, cuyo esquema se llama INFORMATION_SCHEMA[5].

En resumen, el proceso de migración sigue un recorrido arborescente, comenzando por aquellas tablas que no contengan clave foránea, es decir, no tienen dependencias a otras. Estas son hojas de la estructura de tipo arbórea que forman las tablas y sus relaciones. En cada nivel del árbol se asegura que las dependencias fueron cumplidas y por lo tanto es posible migrar el nivel, hasta llegar a la o las raíces, que en general corresponden a relaciones del tipo N:M.

IV. Experimentación

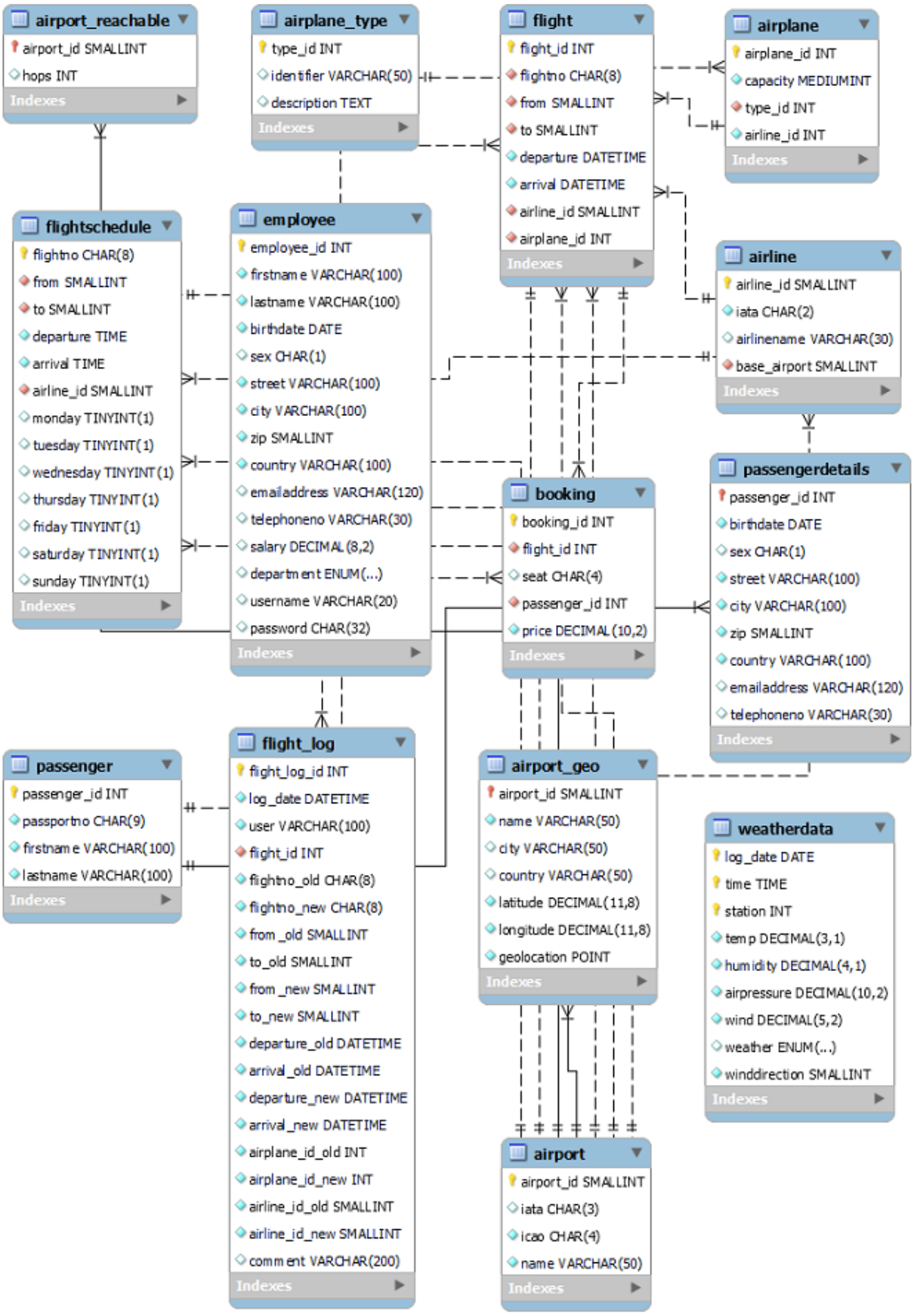
Las pruebas se realizaron utilizando dos bases de datos diferentes y disponibles en el sitio web de MySQL: Sakila[6] y Airportdb[7] (las figuras 1 y 2 muestran cada uno de los diagramas Modelo Entidad-Relación, respectivamente).



Fuente: https://www3.ntu.edu.sg/home/ehchua/programming/sql/images/SampleSakila.png

Figura 1: Diagrama Modelo Entidad-Relación de base de datos Sakila.

[5]https://dev.mysql.com/doc/refman/8.0/en/information-schema.html
[6]https://dev.mysql.com/doc/sakila/en/
[7]https://dev.mysql.com/doc/airportdb/en/airportdb-introduction.html



Fuente: https://dev.mysql.com/doc/airportdb/en/images/airportdb-schema.png

Figura 2: Diagrama Modelo Entidad-Relación de base de datos Airportdb.

Sakila se logró migrar en un tiempo de 6 minutos, generando 14 colecciones. Mediante esta prueba se detectó que el algoritmo no contemplaba los escenarios en los cuales una tabla tiene una o dos CFs y más de dos referencias. Se decidió que estos escenarios sean migrados como colección. El cuadro III muestra las tablas en MySQL y su correspondiente cardinalidad. En la figura 3 se exponen capturas de algunos resultados luego de la migración explicando el patrón aplicado.

Cuadro III: Cardinalidad por tabla en base de datos Sakila.

Nombre de la tabla	***Cantidad de registros***
actor	200
address	603
category	16
city	600
country	109
customer	599
film	1.000
film_actor	5.462
film_category	1.000
film_text	1.000
inventory	4.581
language	6
payment	16.086
rental	16.005
staff	2
store	2

```
_id: ObjectId('62c0a2157ce6797c75d4eaec')
customer_id: 1
store_id: 1
first_name: "MARY"
last_name: "SMITH"
email: "MARY.SMITH@sakilacustomer.org"
address_id: 5
active: true
create_date: "2006-02-14 22:04:36.0"
last_update: "2006-02-15 04:57:20.0"
```


(a) Ejemplo de documento en MongoDB correspondiente a la tabla *customer*, la cual fue migrada como colección.


```
_id: ObjectId('62c0a2217ce6797c75d50319')
rental_id: 1
rental_date: "2005-05-24 22:53:30.0"
inventory_id: ObjectId('62c0a2167ce6797c75d4f299')
customer_id: ObjectId('62c0a2167ce6797c75d4eb6d')
return_date: "2005-05-26 22:04:30.0"
staff_id: ObjectId('62c0a2177ce6797c75d50310')
last_update: "2006-02-15 21:30:53.0"
```


(b) Corresponde a la migración de la tabla *rental*, donde se usó el patrón modelo de enlace.


```
_id: ObjectId('62c0a2147ce6797c75d4e4ee')
actor_id: 1
first_name: "PENELOPE"
last_name: "GUINESS"
last_update: "2006-02-15 04:34:33.0"
films: Array
  0: Object
    _id: ObjectId('62c0a2167ce6797c75d4ed43')
    film_id: 1
    title: "ACADEMY DINOSAUR"
    description: "A Epic Drama of a Feminist And a Mad Scientist w
    release_year: "2006-01-01 00:00:00.0"
    language_id: 1
    original_language_id: null
    rental_duration: 6
    rental_rate: " 0"
    length: 86
    replacement_cost: " 86"
    rating: "PG"
    special_features: "Deleted Scenes,Behind the Scenes"
    last_update: "2006-02-15 05:03:42.0"
  1: Object
  2: Object
  3: Object
  4: Object
  5: Object
  6: Object
  7: Object
  8: Object
  9: Object
  10: Object
  11: Object
```


(c) Corresponde a la migración de la tabla *actor*, que junto a la figura 3d muestran un ejemplo de patrón "embebido en dos sentidos". En Sakila no se tiene casos de aplicación de patrón "embebido en un sentido", pero es muy similar al mostrado en esta imagen.

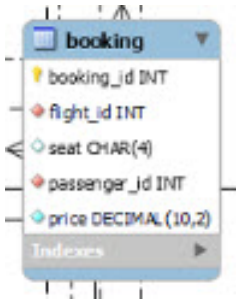

```
_id: ObjectId('62c0a2167ce6797c75d4ed43')
film_id: 1
title: "ACADEMY DINOSAUR"
description: "A Epic Drama of a Feminist And a Mad Scientist who m
release_year: "2006-01-01 00:00:00.0"
language_id: 1
original_language_id: null
rental_duration: 6
rental_rate: " 0"
length: 86
replacement_cost: " 86"
rating: "PG"
special_features: "Deleted Scenes,Behind the Scenes"
last_update: "2006-02-15 05:03:42.0"
actors: Array
  0: Object
    _id: ObjectId('62c0a2147ce6797c75d4e4ee')
    actor_id: 1
    first_name: "PENELOPE"
    last_name: "GUINESS"
    last_update: "2006-02-15 04:34:33.0"
  1: Object
  2: Object
  3: Object
  4: Object
  5: Object
  6: Object
  7: Object
  8: Object
  9: Object
categorys: Array
  0: Object
    _id: ObjectId('62c0a2157ce6797c75d4e5bb')
    category_id: 6
    name: "Documentary"
    last_update: "2006-02-15 04:46:27.0"
```


(d) Corresponde a la migración de la tabla *film*, que junto a la figura 3c muestran un ejemplo de patrón "embebido en dos sentidos".

Figura 3: Ejemplos de documentos como resultado de la migración de Sakila a MongoDB.

La BD Airportdb se logró migrar en un tiempo de 13h 23m, generando 9 colecciones. Aquí se debió reducir el tamaño de la tabla *booking*, ya que dado el volumen original, la herramienta insumía un tiempo que no era razonable de esperar (se pasó de 50 millones de registros a 70 mil). El resto de las tablas quedaron con el tamaño original indicado en el cuadro IV. Mediante la migración de esta BD se detectó que el patrón "embebido en dos sentidos" no contempla, en las tablas unión, el resto de los campos que no son CFs, por lo que se pierde información. Por eso decidimos que para el modelo "embebido en dos sentidos", además de embeber los documentos, también agregar en ambos lados el resto de los campos que el algoritmo no considera.

Durante la migración de esta base se encontraron casos no contemplados que no surgieron en la migración de Sakila. Puntualmente en la migración de la tabla *booking*, que implica aplicar el patrón de "embebido en dos sentidos", donde *passenger* es embebido en *flight* y viceversa, pero se pierden los campos *seat* y *price* de la tabla *booking*. Siendo así, se tomó la decisión para nuestra solución, al embeber *passenger* en *flight*, también se agregar *seat* y *price*. Y lo mismo en el sentido inverso. Este caso se muestra en la figura 4.



(a) Tabla booking en MySQL.

```
_id: ObjectId('62c4edf4876e97444703e995')
flight_id: 3863
flightno: "SP3028"
from: ObjectId('62c4e439876e97444702dd5d')
to: ObjectId('62c4e439876e97444702db73')
departure: "2015-06-01 14:53:00.0"
arrival: "2015-06-02 05:43:00.0"
airline_id: ObjectId('62c4e449876e974447037495')
airplane_id: ObjectId('62c4e449876e9744470374ae')
passengers: Array
  0: Object
     _id: ObjectId('62c4e43d876e97444702f2bd')
     passenger_id: 2947
     passportno: "P105957"
     firstname: "John"
     lastname: "Breaux"
   > passengerdetailss: Array
     booking_id: "1"
     seat: "'null'"
     price: "' 110'"
  1: Object
     _id: ObjectId('62c4e443876e9744470338888')
     passenger_id: 20814
     passportno: "P123824"
     firstname: "Otis"
     lastname: "Pitts"
   > passengerdetailss: Array
     booking_id: "2"
     seat: "'25F'"
     price: "' 173'"
  2: Object
  3: Object
```

(b) Documento en colección *flight*, como resultado de la migración usando "embebido en dos sentidos".

Figura 4: Ejemplo de caso particular en la migración de tabla *booking* en Airportdb, usando "embebido en dos sentidos".

Por último, se detectó un problema al momento de migrar campos de fecha con determinados formatos, dado que el driver JDBC de MySQL no logra interpretar el tipo de dato correctamente. Investigando se llegó a que corresponde a un bug

Cuadro IV: Cardinalidad por tabla en base de datos Airportdb.

Nombre de la tabla	***Cantidad de registros***
booking	50.831.531
flight	41.6429
flight_log	0
airport	9.939
airport_reachable	0
airport_geo	9.854
airline	113
flightschedule	9.851
airplane	5.583
airplane_type	342
employee	1.000
passenger	36.346
passengerdetails	37.785
weatherdata	4.626.432

conocido[8], por lo que se eliminaron las columnas problemáticas, ya que está fuera del alcance de este trabajo. Esto ocurrió con las columnas *birthdate* en las tablas *employee* y *passengerdetails*, de la base de datos Airport.

V. Conclusiones y trabajo futuro

El algoritmo descrito en el artículo provee una correcta migración para bases de datos similares a la utilizada por los autores y con un volumen de datos bajo a medio. Sin embargo, aplicar el algoritmo a estructuras de bases de datos más representativas a soluciones productivas, se comienzan a ver defectos como los detectados en las pruebas. Si vamos al caso de Sakila, la base obtenida mediante el algoritmo es casi similar a la base relacional, a menos de unas pocas colecciones, lo que no agrega demasiado valor a la migración. Si la comparamos contra la base documental en [Harrison], se puede ver que aplicando patrones de diseño es posible mejorar notoriamente y ajustarla al caso de uso que tendrá Por lo tanto, entendemos que este algoritmo sirve como punto de partida, pero requiere un trabajo a futuro de análisis, que permita interpretar mejor la base de datos origen, considerando estadísticas de lecturas y escrituras, para aplicar patrones de diseño que generen como resultado una base de datos documental más acorde a la realidad en la cual va a ser utilizada.

Otro punto a incorporar en el algoritmo, es la migración de índices y procedimientos almacenados que estén incluidos en la base relacional. Los índices pueden ser agregados a través de opciones al momento de crear las colecciones y los procedimientos almacenados pueden migrarse como *Database Events*.

En cuanto a la herramienta desarrollada, existen varios aspectos de mejora desde el punto de vista de rendimiento que se deben considerar si se quiere apuntar a bases de datos de grandes volúmenes, como es el ejemplo de la base Airportdb, la cual debimos acotar. Esto podría realizarse mediante una mejora en la carga de datos en MongoDB, ya sea mediante estructuras temporales intermedias, o la generación de índices que permitan una lectura de los datos ingresados en MongoDB de forma más eficiente. Por el lado de la lectura de datos en MySQL no detectamos aspectos de mejora, ya que el proceso es eficiente. Esto se pudo observar en la migración de las tablas *weatherdata* y *booking* de la base Airportdb, en donde, si bien la primer tabla es mucho mayor a la segunda en cuanto a cantidad de registros (4 millones contra 70 mil en nuestro experimento), el tiempo de migración de la primer tabla fue de 30 minutos aproximadamente, mientras que la segunda tabla insumió unas 9 horas. La gran diferencia se da ya que la primer tabla fue migrada simplemente como colección, mientras que la segunda implicó utilizar el método “embebido en dos sentidos”, lo que requiere acceso a las colecciones ya migradas en MongoDB.

Otro aspecto de mejora está relacionado con el uso de Kettle y sus funcionalidades de trabajos, aumento en el uso de hilos de procesamiento, etc, que no fue posible en esta instancia dado que requiere un aprendizaje mayor de la herramienta y no es parte del objetivo actual de este trabajo.

[8]https://bugs.mysql.com/bug.php?id=82005

REFERENCIAS

Bhat, U. and Jadhav, S. (2010). Volume 1 – no. 13 moving towards non-relational databases.

[Harrison]. Sakila sample schema in mongodb. http://guyharrison.squarespace.com/blog/2015/3/23/sakila-sample-schema-in-mongodb.html.

[MongoDB1]. Database References — Manual References. https://www.mongodb.com/docs/manual/reference/database-references/#std-label-document-references.

[MongoDB2]. Model One-to-One Relationships with Embedded Documents. https://www.mongodb.com/docs/manual/tutorial/model-embedded-one-to-one-relationships-between-documents/.

[MongoDB3]. Model One-to-Many Relationships with Embedded Documents. https://www.mongodb.com/docs/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/.

Stanescu, L., Brezovan, M., and Burdescu, D. D. Automatic mapping of MySQL databases to NoSQL MongoDB. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 837–840.