Aplicando diferentes metodologías al diseño de bases de datos de documentos

Juan Ignacio Betarte
Instituto de Computación
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
juan.betarte@fing.edu.uy

Daniela Andrade
Instituto de Computación
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
daniela.andrade@fing.edu.uy

***Resumen*—El rendimiento de las bases de datos no relacionales está muy atado a su diseño. Al no disponer de una única forma de trabajar, se plantea realizar una comparación entre diseños basados en diferentes metodologías para evaluar cuál se adecúa más a una realidad específica. Las metodologías utilizadas fueron: las heurísticas de Hoberman y la propuesta en la documentación de MongoDB. Se realizaron pruebas experimentales en base a una realidad planteada, donde los resultados mostraron muy poca discrepancia tanto en los esquemas diseñados como en su rendimiento.**

I. Introducción

Hoy en día, el rendimiento de las bases de datos no relacionales dependen mucho de su diseño. Es por este motivo, que nuestra investigación se centra en comparar el rendimiento entre dos diseños de bases de datos no relacionales, en particular bases de datos de documentos.

Para llevar a cabo nuestra investigación, consideraremos como problema a resolver una aplicación de películas, donde los usuarios pueden visualizar la información de las mismas y además, pueden agregar valoraciones. Como principal objetivo, analizaremos dos metodologías para el diseño de bases de datos, por un lado la documentación de MongoDB y por otro, las heurísticas sugeridas por Hoberman de si incrustar o hacer referencias a documentos. Llevaremos a cabo el diseño de las mismas en base a la realidad planteada para así, poder evaluar el desempeño y concluir cuál de los dos se adapta mejor al problema.

El presente documento se organiza de la siguiente manera, en la seccción dos se presentan los trabajos relacionados. En la sección tres, se describen las metodologías de MongoDB y de Hoberman. En la sección cuatro, se plantea la descripción del problema a resolver. En la sección cinco, se describe el diseño del esquema de las bases de datos. En la sección seis, se describe la carga de los datos, la implementación de las consultas utilizadas para las pruebas, la configuración del ambiente y los resultados de las pruebas. Por último, en la sección siete se presenta el análisis de los resultados, los problemas enfrentados y las conclusiones.

II. Trabajos relacionados

Debido a que el rendimiento de las operaciones sobre base de datos de documentos dependen mucho del diseño de estas, es necesario realizar el diseño adecuado al problema planteado. Para esto, analizamos las heurísticas propuestas por Hoberman en NoSQL Databases and Data Modeling Techniques for a Document-oriented NoSQL Database [3] y Data Modeling for MongoDB [2]. Y por otro lado, la documentación de MongoDB[1].

También, analizamos el artículo Mortadelo: A Model-Driven Framework for NoSQL Database Design [1].

III. Metodologías

III-A. Sugerencias de la documentación de MongoDB

Al ser, principalmente, las necesidades de nuestra aplicación operaciones de lectura en una colección, agregar índices puede mejorar el rendimiento.

Al momento de diseñar el esquema de una base de datos, es necesario decidir si incrustar o hacer referencias a documentos.

III-A1. Modelos de datos integrados: Estos esquemas se conocen como modelos desnormalizados. Permiten que se almacene información relacionada en el mismo documento, esto reduce la cantidad de consultas o actualizaciones que se deben realizar para completar una operación. Se recomienda utilizar modelos de datos integrados cuando:

- Se tienen relaciones de uno a uno entre entidades. Para este diseño se pueden utilizar dos patrones, el patrón de documentos incrustados o el patrón de subconjuntos. En el primero de ellos, se mantiene toda la información de uno de los documentos dentro del otro, pero esto puede tener como inconveniente que se generen documentos grandes que contienen información que la aplicación no necesite o que se consulten con poca frecuencia. Por este motivo, se puede utilizar el segundo patrón, que implica tener incrustada dentro de la colección la información que la aplicación necesita con más frecuencia, y el resto en otra colección. Este segundo patrón mejora el rendimiento de las operaciones de lectura. Pero puede dificultar a la hora de hacer mantenimiento de la base de datos[2].

[1] https://docs.mongodb.com/manual/

[2] https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-one-relationships-between-documents/#std-label-data-modeling-example-one-to-one

- Se tienen relaciones de uno a varios entre entidades. Para este diseño se pueden utilizar dos patrones, el patrón de documentos incrustados o el patrón de subconjuntos. En el primero de ellos, se mantiene toda la información de uno de los documentos dentro de un arreglo del otro. Esto tiene el mismo inconveniente que el caso anterior, agregando el caso que el campo incrustado no esté acotado. Por este motivo, se puede utilizar el segundo patrón, que implica mantener incrustados aquellos datos que se acceden con frecuencia por la aplicación, y el resto en otra colección. Este segundo patrón mejora el rendimiento de las operaciones de lectura. Pero puede dar como resultado la duplicación de los datos y dificultar el mantenimiento de la base de datos[3].

III-A2. Modelos de datos normalizados: Los modelos de datos normalizados describen relaciones utilizando referencias entre documentos. Estos tipos de modelos en general se utilizan cuando:

- Cuando la incrustación da lugar a datos duplicados y no proporciona las ventajas de rendimiento de lectura para compensar el costo de los datos duplicados.
- Cuando se quiere representar relaciones de varios a varios.
- Cuando se quiere modelar grandes conjuntos de datos jerárquicos.

En estos tipos de modelos, es posible evitar la repetición de los datos. Además, en relaciones uno a varios, es posible elegir en que colección almacenar la referencia, según sea conveniente.

III-A3. Otras consideraciones: Algunos factores a tener en cuanta[4]:

- Crear índices para los campos que son consultados con frecuencia y en aquellos que devuelven resultados ordenados. Esto es beneficioso en los casos donde las operaciones más comunes son de lectura. Hay que tener en cuenta el consumo de espacio en disco y de memoria.
- Tener una gran cantidad de colecciones no significa tener un menor rendimiento, esto depende mucho de las relaciones entre los datos. Si se agrupan los documentos por tipo, podría ser útil mantener una colección por cada tipo.
- Al mantener documentos incrustados, se debe considerar mover los campos comunes al documento padre para mantener menos copias y reducir el costo de mantener el índice para estos.
- Utilizar nombres de campos cortos reduce la memoria utilizada, especialmente cuando los documentos son pequeños.

III-B. Metodología propuesta por Hoberman

Hoberman en su libro [2] expone una guía sobre cómo enfocar el modelado en bases de datos documentales, particularmente para MongoDB.

[3]https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/#std-label-data-modeling-example-one-to-many

[4]https://docs.mongodb.com/manual/core/data-model-operations/

El modelado dispone distintos niveles de granularidad (Fig. 1). Se comienza con el Modelado de Datos Conceptual (CDM), donde se captura un panorama general de los requerimientos del sistema a modelar. Aquí sugiere realizar cinco pasos:

- Realizarse las siguientes preguntas:
  - ¿Qué hará la aplicación?
  - ¿"cómo es" o "cómo será"? (Respecto al ambiente de negocio)
  - ¿Realizar análisis sobre los datos es un requerimiento?
  - ¿Quién es la audiencia?
  - ¿Flexibilidad o simplicidad?
- Determinar los conceptos claves (entidades).
- Capturar las relaciones entre las entidades.
- Determinar el mejor modelo.
- Verificar y confirmar.

Luego de finalizado el modelado conceptual, sugiere realizar el Modelado Lógico (LDM), y su principal objetivo es detallar la solución sobre la realidad planteada.
Consiste en los siguientes pasos:

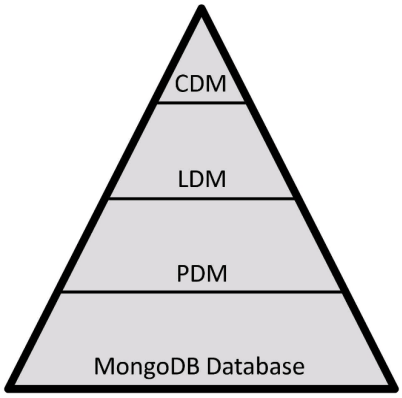

Figura 1: Niveles de diseño

- Rellenar el CDM.
- Normalizar y abstraer.
- Determinar el modelo mas adecuado.
- Verificar y confirmar.

Finalmente, se realiza el Modelado Físico de Datos (PDM), el cual consiste en obtener una solución técnica detallada de la realidad. Es aquí que se toma realmente en cuanta la tecnología y los problemas que presenta, como el almacenamiento, la seguridad y el rendimiento.
Los principales objetivos son:

- ¿Cómo son las colecciones?
- ¿Cuál es la forma más óptima de hacer fragmentación?
- ¿Cómo podemos hacer la información más segura?
- ¿Cómo podemos responder esta pregunta de negocio en menos de 300 milisegundos?

Cómo se explicó en la metodología anterior para el modelado de relaciones en MongoDB, existen dos maneras, embeber información o referenciar otros documentos. Según Hoberman, embeber es el proceso de mezclar dos o más entidades en una

sola, usando una jerarquía. Y referenciar, es parecido a crear una clave foránea en una base de datos relacional, que sirve como un puntero de una entidad a otra. Hoberman sugiere cinco heurísticas para tomar la decisión sobre qué es mejor, referenciar o embeber.

1. La información que es frecuentemente consultada desde varias entidades (colecciones) al mismo tiempo, puede ser embebida en un solo documento.
2. Entidades que son consideradas entidades débiles, pueden ser embebidas en una sola entidad.
3. Si existe una relación uno a uno entre dos entidades, se embebe una entidad dentro de la otra.
4. Las entidades que presentan volatilidad (inserciones, actualizaciones y eliminaciones) con la misma frecuencia pueden estar embebidas dentro de una entidad.
5. Entidades que no son claves, pero tienen relaciones con entidades que sí son claves, pueden estar referenciadas en vez de embebidas.

IV. Caso de estudio

Para llevar a cabo nuestra investigación, consideraremos una aplicación de películas, inspirada en las plataformas de streaming, donde se tendrá una vista previa de veinte películas elegidas al azar, donde se podrá visualizar para cada una el título, una descripción, el género, la duración, la fecha de estreno, los idiomas hablados y si es para personas mayores de dieciocho años. Además, se podrá visualizar la información completa seleccionando alguna de ellas.

Un usuario de la aplicación, podrá realizar valoraciones a las diferentes películas, así como también, realizar búsquedas utilizando ciertos filtros, como ser, idiomas hablados, edad, valoración, género, fecha de estreno, entre otros.

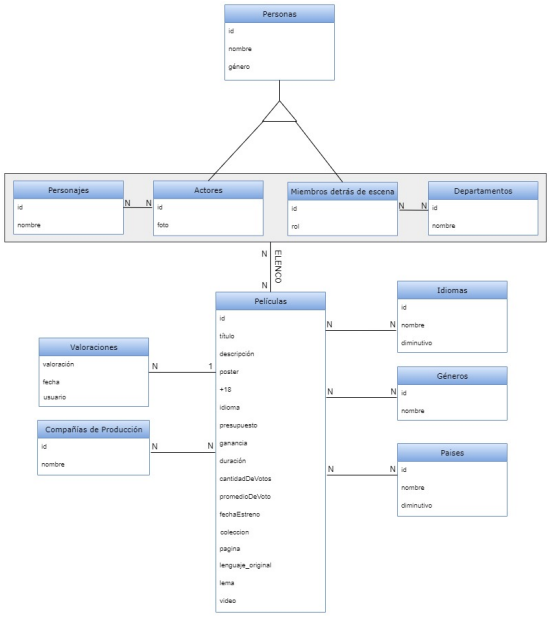

Figura 2: Caso de estudio

En el Cuadro I se especifican las consultas y operaciones que se realizarán en la aplicación, junto con su frecuencia.

Nro.	***Operación***	***Frecuencia por segundo***
1	Obtener la información básica de veinte películas aleatoriamente (título, descripción, género, duración, año, idiomas, edad)	100
2	Obtener detalle sobre la información de una película	60
3	Obtener todas las películas que son del género Comedia, con valoración mayor o igual a tres	30
4	Obtener el elenco de una película	20
5	Obtener todos los usuarios que realizaron valoraciones para cierta película	60
6	Agregar valoración	50

Cuadro I: Consultas y operaciones

V. Diseño

V-A. *JSON Schema de MongoDB y decisiones tomadas*

Se definen tres colecciones de acuerdo a la realidad planteada y las recomendaciones de la documentación de MongoDB.

Debido a que en nuestra realidad, son más frecuentes las operaciones de consulta que de actualización, se consideran las recomendaciones realizadas en la sección III-A1. Debido a que en la pantalla de inicio no se muestra toda la información relacionada a las películas, se decidió incluir en la colección ”Pelicula” los atributos que se acceden con mayor frecuencia.

```
$jsonSchema: {
   "bsonType": "object",
   "title": "Pelicula",
   "required": ["titulo", "fecha", "descripcion",
   "idioma", "duracion", "genero", "adulto"],
   "properties": {
      "_id": { "bsonType": "objectId" },
      "titulo": { "bsonType": "string" },
      "fecha": {"bsonType": "string" },
      "descripcion": { "bsonType": "string" },
      "+18": { "bsonType": "string" },
      "idioma": {
         "bsonType": "array",
         "items": {
            "bsonType": "object",
            "required": ["nombre", "diminutivo"],
            "properties": {
               "nombre": { "bsonType": "string" },
               "diminutivo": { "bsonType": "string" }
```

```
            }
         },
         "additionalItems": false,
         "minItems": 1,
         "uniqueItems": true
      },
      "duracion": { "bsonType": "string" },
      "genero": {
         "bsonType": "array",
         "items": {
            "bsonType": "object",
            "required": ["nombre"],
            "properties": {
               "nombre": { "bsonType": "string" }
            }
         },
         "additionalItems": false,
         "minItems": 1,
         "uniqueItems": true
      }
   }
}
```

El resto de la información, se incluyó en otra colección denominada ”Detalle_Pelicula”. Donde se decidió incluir la información del elenco incrustada en la colección, si bien, la documentación de MongoDB recomienda hacer referencia a documentos cuando se quiere representar relaciones varios a varios, en nuestro caso, los documentos de los actores y los de detrás de escena tienen dos y tres atributos del tipo texto y entero respectivamente. Mantener la información incrustada no implica inconvenientes debido a que no se realizan actualizaciones de los datos luego que se da de alta la película. Además, la información de estos sólo se consulta cunado se consulta la información de la película. Pero por el lado del resto de las operaciones, hacer referencia implicaría mantener dos colecciones, una para mantener el nombre del actor y otra para mantener los personajes en las diferentes películas. Lo mismo ocurriría con los detrás de escena. Esto podría elevar los tiempos a la hora de consultar la información.

```
$jsonSchema: {
   "bsonType": "object",
   "title": "Detalles_Peliculas",
   "required": ["pelicula_id", "poster", "compania_prod",
    "presupuesto", "ganancia", "pais", "cant_votos",
    "promedio_votos", "coleccion", "pagina",
    "lenguaje_original", "lema", "video", "valoracion"],
   "properties": {
      "_id": { "bsonType": "objectId" },
      "idPelicula": {
         "bsonType": "objectId",
         "ref": "Pelicula"
      },
      "poster": { "bsonType": "string" },
      "compania_prod": {
         "bsonType": "array",
         "items": {
            "bsonType": "object",
            "required": ["nombre"],
            "properties": {
               "nombre": { "bsonType": "string" }
            }
         },
         "additionalItems": false,
         "minItems": 1,
         "uniqueItems": true
      },
      "presupuesto": { "bsonType": "string" },
      "ganancia": { "bsonType": "string" },
      "pais": {
         "bsonType": "array",
         "items": {
            "bsonType": "object",
            "required": ["nombre"],
            "properties": {
               "nombre": { "bsonType": "string" }
            }
         },
         "additionalItems": false,
         "minItems": 1,
         "uniqueItems": true
      },
      "cant_votos": { "bsonType": "string" },
      "promedio_votos": { "bsonType": "string" },
      "pagina": { "bsonType": "string" },
      "lenguaje_original": { "bsonType": "string" },
      "lema": { "bsonType": "string" },
      "video": { "bsonType": "string" },
      "detras_escena": {
         "bsonType": "array",
         "items": {
            "bsonType": "object",
            "required": ["nombre", "rol", "departamento"],
            "properties": {
               "nombre": { "bsonType": "string" },
               "rol": { "bsonType": "string" },
               "departamento": {"bsonType": "string" }
            }
         },
         "additionalItems": false,
         "minItems": 1,
         "uniqueItems": true
      },
      "actores": {
         "bsonType": "array",
         "items": {
            "bsonType": "object",
            "required": ["nombre", "parsonaje"],
            "properties": {
               "nombre": { "bsonType": "string" },
               "parsonaje": { "bsonType": "string" }
            }
         },
         "additionalItems": false,
         "minItems": 1,
         "uniqueItems": true
      }
   }
}
```

Por último, considerando las recomendaciones de la sección III-A2, se mantiene en la colección ”Valoraciones”, las valoraciones que realizan los usuarios a las diferentes películas, representando una relación varios a varios con la colección ”Pelicula”. Además, es una colección en la que se hacen inserciones con frecuencia y su crecimiento puede elevar los tiempos de respuesta en el resto de las colecciones si se mantiene incrustada en alguna.

```
$jsonSchema: {
   "bsonType": "object",
   "title": "Valoraciones",
   "bsonType": "object",
   "required": ["pelicula_id", "valoracion",
   "fecha", "idUsuario"],
   "properties": {
      "_id": { "bsonType": "objectId" },
      "idPelicula": {
         "bsonType": "objectId",
         "ref": "Pelicula"
      },
      "valoracion": { "bsonType": "string" },
      "fecha": {"bsonType": "string" },
      "idUsuario": {"bsonType": "string" }
   }
}
```

V-B. JSON Schema utilizando la metodología propuesta por Hoberman

Como se mencionó anteriormente en la realidad propuesta son mucho más frecuente las operaciones de lectura que las de escrituras. Por lo tanto, toma bastante importancia la heurística I.

Analizando las consultas, vemos que la que se realiza con mayor frecuencia es: **Obtener la información de películas (título, descripción, género, duración, año, idiomas, edad)**. En esta consulta, las entidades que se ven involucrados son: Películas, Idiomas y Géneros. Por lo tanto, resulta conveniente representar las tres entidades en un documento. Como Película es la entidad clave, es preferible embeber Idiomas y Géneros dentro de la misma. La colección resultante la denominaremos Película.

```
$jsonSchema: {
      "bsonType": "object",
      "title": Pelicula,
      "required": ["titulo", "fecha","+18","duracion"],
      "properties": {
            "_id": {"bsonType": ObjectId},
            "titulo": {"bsonType": string},
            "descripcion": {"bsonType": string},
            "poster": {"bsonType": string},
            "+18": {"bsonType": string},
            "fecha": {"bsonType": string},
            "promedio_votos": {"bsonType": string},
            "cant_votos": {"bsonType": string},
            "duracion": {"bsonType": string},
            "poster": {"bsonType": string},
            "lenguaje_original": { "bsonType": "string" },
            "lema": { "bsonType": "string" },
            "video": { "bsonType": "string" },
            "generos": {
                  "bsonType": "array"
                  "items": {
                        "bsonType": "object",
                        "required": {"nombre"},
                        "properties": {
                              "nombre" : {"bsonType": "string"}
                        }
                  }
            },
            "idiomas": {
                  "bsonType": "array"
                  "items": {
                        "bsonType": "object",
                        "required": {"nombre"},
                        "properties": {
                              "nombre" : {"bsonType": "string"},
                              "diminutivo": {"bsonType": "string"}
                        }
                  }
            }
      }
}
```

Luego, la segunda consulta más frecuente es cuando se quiere obtener información adicional de cierta película. Dicha consulta incluye obtener el Elenco correspondiente, por lo tanto, parece apropiado que estén embebidas dentro de un mismo documento. Por otro lado, el resto de la información asociada también se embeben dentro de un mismo documento. Esto incluye las entidades Compañías de Producción, Países y Película. La pregunta que surge aquí es si embeber el Elenco junto con el resto de información de la película. En este caso, continuando con el lineamiento de la heurística I, notando que presentan la misma Volatilidad (III), y que rara vez se obtiene el Elenco de una Película sin mostrar el resto de la información, se decidió que quedé todo embebido dentro de un mismo documento, logrando una colección que denominaremos Detalles_Peliculas.

```
$jsonSchema: {
      "bsonType": "object",
      "title": Detalles_Peliculas,
      "required": ["idPelicula","elenco"],
      "properties": {
            "_id": {"bsonType": ObjectId},
            "idPelicula": {
                  "bsonType": ObjectId,
                  "ref": Peliculas
            },
            "compania_prod": {
                  "bsonType": "array"
                  "items": {
                        "bsonType": "object",
                        "required": {"nombre"},
                        "properties": {
                              "nombre" : {"bsonType": "string"}
                        }
                  }
            },
            "pais": {
               "bsonType": "array",
               "items": {
                  "bsonType": "object",
                  "required": ["nombre"],
                  "properties": {
                     "nombre": { "bsonType": "string" }
                  }
               },
               "additionalItems": false,
               "minItems": 1,
               "uniqueItems": true
            },
            "elenco": {
               "bsonType": "object",
               "title": "elenco"
               "required": {actores,destras_escena},
               "properties": {
                     "actores": {
                           "bsonType": "array",
                           "items": {
                           "bsonType": "object",
                           "required": {"nombre"},
                           "properties": {
                        "nombre" : {"bsonType": "string"},
                        "personaje" : {"bsonType": "string"}
                           }
                           }
                     },
                     "detras_escena": {
                           "bsonType": "array",
                           "items": {
                           "bsonType": "object",
                           "required": {"nombre"},
                           "properties": {
                                 "nombre" :
                                 {"bsonType": "string"},
                                 "rol" :
                                 {"bsonType": "string"},
                                 "departamento" :
                                 {"bsonType": "string"}
                           }
                           }
                     }
            }
            }
      }
}
```

Finalmente, sólo falta modelar las valoraciones realizadas por los usuarios, las cuales serán incluidas dentro de una colección denominada Valoraciones. Principalmente, debido a que no hay un consulta específica que obtenga las valoraciones, y además, presenta distinta volatilidad que el resto de las colecciones.

```
$jsonSchema: {
   "bsonType": "object",
   "title": "Valoraciones",
   "bsonType": "object",
   "required": ["idPelicula", "valoracion",
   "idUsuario", "fecha"],
   "properties": {
      "_id": { "bsonType": "objectId" },
      "idPelicula": {
         "bsonType": "objectId",
         "ref": "Pelicula"
      },
      "idUsuario": {"bsonType": "string" },
      "valoracion": { "bsonType": "string" },
      "fecha": {"bsonType": "string" }
   }
}
```

VI. Experimentación

VI-A. Carga de Datos

Se extrajo los datos obtenidos sobre películas [5], para poder cargar las bases de datos correspondientes a los esquemas obtenidos anteriormente. Para realizar este labor [6], se utilizó Python 3.9.6 para acceder a tres archivos en formato JSON (credits.json, ratings.json y moviesmetadata.json) y obtener los datos deseados, para finalmente, generar los documentos con la estructura adecuada para insertarlos en la colección correspondiente.

VI-B. Consultas de prueba

A continuación se listan las consultas utilizadas, implementadas en el lenguaje Groovy.

- Consulta 1: Obtener la información básica de veinte películas aleatoriamente (título, descripción, género, duración, año, idiomas, edad).

```
MongoCollection<Document> collection =
   database.getCollection("Pelicula");
AggregateIterable output = collection.aggregate(
   Arrays.asList(
            sample(20),
            project(
               new Document("titulo",1).
                  append("descripcion",1).
                  append("generos",1).
                  append("duracion",1).
                  append("fecha",1).
                  append("idiomas",1).
                  append("+18",1)
         )
   )
)
```

- Consulta 2: Obtener detalle sobre la información de una película.

```
MongoCollection<Document> collection =
   database.getCollection("Pelicula");
Bson lookup = new Document
   ('$lookup',
      new Document('from', 'Valoraciones' )
      .append('localField', '_id')
      .append('foreignField', 'idPelicula')
      .append('as', 'valoraciones')
   );
Bson lookup2 = new Document
   ('$lookup',
            new Document('from', 'Detalles_Peliculas')
            .append('localField', 'idPelicula')
            .append('foreignField', 'idPelicula')
            .append('as', 'detalle_pelicula')
      );
AggregateIterable output = collection.aggregate(
   Arrays.asList(
      match(
         new Document(
            "_id",
            new ObjectId("60f5cf04a11585566ce3c62b")
         )
      ),
      lookup,
      lookup2,
         project(
            new Document("titulo",1).
                  append("descripcion",1).
                  append("generos",1).
                  append("duracion",1).
                  append("fecha",1).
                  append("idiomas",1).
                  append("+18",1).
                  append('generos',1).
                  append("valoraciones.valoracion", 1).
                  append("detalle_pelicula.actores",1).
                  append("detalle_pelicula.detras_escena",1)
         )
      )
)
```

Para el caso de Hoberman, la consulta varía en cómo se accede a los atributos del elenco:

```
project(
   new Document("titulo",1).
      append("descripcion",1).
      append("generos",1).
      append("duracion",1).
      append("fecha",1).
      append("idiomas",1).
      append("+18",1).
      append('generos',1).
      append("valoraciones.valoracion", 1).
      append("detalle_pelicula.elenco",1))
)
```

- Consulta 3: Obtener todas las películas que son del género Comedia, con valoración mayor o igual a tres.

```
MongoCollection<Document> collection =
   database.getCollection("Valoraciones");
Bson lookup = new Document
   ('$lookup',
            new Document('from', 'Pelicula' )
            .append('localField', 'idPelicula')
            .append('foreignField', '_id')
            .append('as', 'pelicula')
      );
AggregateIterable output =
   collection.aggregate(
      Arrays.asList(
         match(Filters.gte("valoracion", "3.0")),
            lookup,
            match(
               new Document(
                  'pelicula.generos.nombre',
                  'Comedy'
               )
            )
      )
   )
```

- Consulta 4: Obtener el elenco de una película.

```
MongoCollection<Document> collection =
   database.getCollection("Detalles_Peliculas");
```

[5]https://www.kaggle.com/rounakbanik/the-movies-dataset?select=movies_metadata.csv

[6]https://github.com/dandra12011995/BDNR

```
AggregateIterable output =
   collection.aggregate(
      Arrays.asList(
         match(
            new Document(
               "idPelicula",
               new ObjectId("60f5cf04a11585566ce3c62b")
            )
         ),
         project(
            new Document("actores",1).
            append("detras_escena",1)
         )
      )
   )
```

Para el caso de Hoberman, la consulta varía al acceder a los atributos del elenco:

```
project(
   new Document("elenco",1)
)
```

- Consulta 5: Obtener todos los usuarios que realizaron valoraciones sobre cierta película.

```
MongoCollection<Document> collection =
   database.getCollection("Valoraciones");
Bson group = new Document(
            '$group',
                        new Document('_id', '$idUsuario')
                    );
AggregateIterable output =
   collection.aggregate(
      Arrays.asList(
         match(
            new Document(
               "idPelicula",
               new ObjectId("60f5cf04a11585566ce3c62b"),
            )
         )
         group
      )
   )
```

- Agregar valoraciones.

```
MongoCollection<Document> collection =
   database.getCollection("Valoraciones");
Date hoy = new Date()
Document document =
   new Document(
      "idPelicula",
      new ObjectId("60f5ced7a11585566ce3194b")
   )
      .append("idUsuario", "2")
      .append("valoracion", "4.0")
      .append("fecha", hoy.getTime().toString());

collection.insertOne(document);
```

VI-C. Configuración del ambiente de pruebas

Las pruebas se realizaron en un equipo con las siguientes características:

- Sistema operativo: Windows 10 64 bits
- Procesador: Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz 2.49GHz
- RAM: 16.0 GB

Se instaló las siguientes veersiones de las herramientas:

- MongoDB: 4.4.6 Community
- JMeter: 5.4

VI-D. Pruebas con JMeter

Para llevar a cabo las pruebas de rendimiento, se utilizó JMeter. El objetivo de las pruebas, fue simular las peticiones especificadas en el Cuadro I para cada diseño de base de datos. Para esto, se creó un plan de prueba para cada consulta, y para cada diseño. Este plan, se ejecutó cinco veces.

Nuestro análisis, se centró en el campo promedio, el cual refiere al tiempo promedio en milisegundos que requiere realizar la cantidad de operaciones especificada para cada prueba.

En las Figuras que se presentan a continuación, se muestran los resultados obtenidos de ejecutar las pruebas de desempeño para cada consulta con las especificaciones mencionadas.

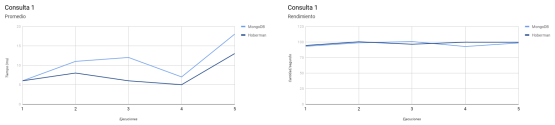

Figura 3: Consulta 1

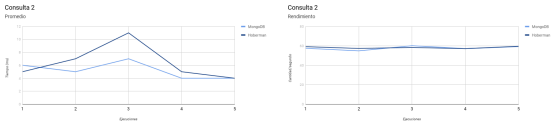

Figura 4: Consulta 2

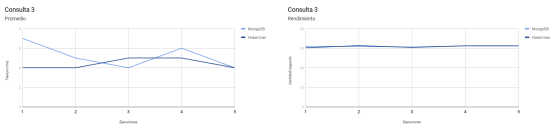

Figura 5: Consulta 3

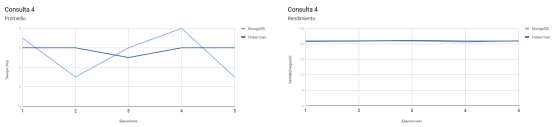

Figura 6: Consulta 4

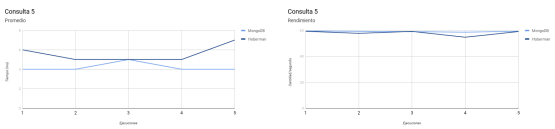

Figura 7: Consulta 5

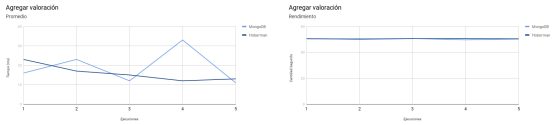

Figura 8: Agregar valoración

VII. Conclusiones y trabajo futuro

Observando los resultados de las pruebas podemos decir que no hay grandes diferencias de rendimiento entre ambos diseños para la realidad planteada. Lo cual es entendible ya que la estructura de los esquemas no presentan grandes diferencias. Si bien el objetivo de ambas metodologías es presentar ciertos lineamientos para facilitar las decisiones del diseñador, dependiendo de la realidad hay ocasiones en la que algunos requerimientos se pueden modelar de distintas maneras. Aquí es donde entra en juego la habilidad del diseñador, quién debe tomar una decisión, o sino deberá probar todas las alternativas para poder determinar la mejor opción. Otro factor que puede haber influido en el resultado es que la realidad presentada no posee gran complejidad y el volumen de datos utilizados no es lo suficientemente grande como para notar diferencias.

Una diferencia que existe entre ambas metodologías es que Hoberman está más enfocado en proporcionar herramientas para entender cada aspecto de la realidad, y lograr modelarlo de la mejor manera. Mientras que Mongo está más enfocado al diseño del esquema en sí.

En cuanto a los problemas enfrentados, el objetivo inicial del informe era comparar el diseño de los esquemas mediante la metodología de Hoberman con el Framework Mortadelo [7]. Pero no se pudo ejecutar con éxito el Framework Mortadelo, surgían errores en tiempo de ejecución y debido al límite de tiempo con el que se contaba se decidió descartar esta opción.

Luego, la carga de los datos en los diferentes esquemas, implicó reiteradas ejecuciones. Tuvimos problemas para configurar el ambiente para JMeter, a la hora de ejecutar las pruebas teníamos errores relacionados a las versiones de las librerías.

Como trabajo a futuro, se plantea crear índices con las recomendaciones de la documentación de MongoDB[8], para comparar los resultados de rendimiento entre usar o no índices. También, se plantea realizar las pruebas en un equipo con mayores recursos. Donde, se puede considerar un mayor conjunto de datos y adaptar los esquemas de ser necesario.

Referencias

[1] Alfonso de la Vega(B), Diego García-Saiz, Carlos Blanco, Marta Zorrilla, and Pablo Sánchez. Mortadelo: A Model-Driven Framework for NoSQL Database Design. , 2018.

[2] S. Hoberman. *Data Modeling for MongoDB*. Technics Publications, 2 Lindsley Road,Basking Ridge, 2014.

[3] Robert T. Mason. NoSQL Databases and Data Modeling Techniques for a Document-oriented NoSQL Database. http://proceedings.informingscience.org/InSITE2015/InSITE15p259-268Mason1569.pdf, 2015.

[7] https://github.com/alfonsodelavega/mortadelo

[8] https://docs.mongodb.com/manual/tutorial/analyze-query-plan/