Máster en Data Science. URJC
Esta es la asignatura

Este es el título.

AUTOR 1, AUTOR 2

1 de abril de 2018

Índice

Lista de Figuras

1. MongoDB

La primera parte de la práctica estará enfocada en la Base de datos (BBDD) no ralacional MongoDB. Esta se trate de la BBDD más importante actualmente. En el momento de escribir este texto, se encuentra en el 5to de BBDD más utilizadas según la web `https://db-engines.com/en/ranking` y solo por detrás de las BBDD relacionales.

MongoDB tiene las siguientes características:

- Es schemas less, es decir, no es preciso definir un esquema de datos. Aunque no por ello, tenemos que dejar de saber como son almacenados los datos.
- Usa el formato BSON para almacenar los datos, haciendo que sea facil su uso desde lenguajes de programación como Python y JavaScript.
- Permite realizar agregaciones. Incluye el framework aggregation el cuál le permite hacer operaciones realmente potentes.
- Podemos crear índices (incluyendo índices parciales) de cualquiera de sus campos, lo que acelera notablemente las busquedas.
- Su característica de *sharding* le permite ser muy escalable.
- Cuenta con una gran comunidad de desarrolladores detrás lo cual es una garantía de futuro.
- y muchas más.

Para la realización de esta parte de la práctica vamos a partir de tres documentos JavaScript Object Notation (JSON), los cuales podemos ver la forma de obtenerlos en el anexo A.

Cada uno de estos documentos contiene la información sobre los tres tipos de publicaciones que vamos a usar:

- Articles.
- Inproceedings.
- Incollections.

1.1. Diseño

Aunque se dice que MongoDB se trata de una BBDD *Schemasless*, eso no quita de definir una correcta estructura para almacenar los datos la cual nos permita que las consultas que se hagan sean tanto más sencillas y más rápidas. También es importante el saber definir correctamente los índices que vamos a aplicar. En este sentido hay que tener en cuenta que el uso de índices aumenta el rendimiento de nuestras consultas de busqueda en la BBDD, pero resulta perjudiciales a la hora de hacer inserciones, además de ocupar espacio en memoria.

También es importante a la hora de definir la forma en la que se guardarán los datos el hecho de como gestionar las relaciones entre documentos. A grandes ragos, y sin entrar en muchos detalles, existen dos estrategias:

- Utilizar referencias entre documentos. Es decir, que uno de los campos de un documento almacene un identificador que sirva para identificar unívocamente a otro documento. Esta relacion puede darse en ambos sentidos. La principal ventaja es el ahorro de espacio al no tener elementos duplicados, así como la facilidad de gestionar una posible actualización de los datos. Por contra, las queries de busqueda serán más complejas al tener que también hacer una busqueda por el identificador del segundo documento.
- El segundo método consite en insertar el segundo documento dentro del primero como si de un campo más se tratase. Esto Facilita las queries de busqueda ya que recuperamos ambos documentos a la vez, pero tiene la pega de que muy probablemente dupliquemos datos lo cuál se traduce en una mayor cantidad de datos a almacenar y hace más complejo la actualización.

El hecho de optar por una u otra estrategia dependerá de las operaciones que vayamos a realizar sobre nuestra BBDD.

Para el caso que nos ocupa, hemos decidido crear una colección por cada tipo de publicación que vamos a trabajar. También vamos a crear una cuarta colección dentro de la cual cada documento representará un autor. Además, cada uno de estos podocumentos almacenara una información mínima de las publicaciones de dicho autor con el fín de facilitar las queries futuras. Además, se considera que la información de un libro, y la relación de este con sus autores, no debería sufrir cambios (salvo casos excepcionales), por lo que el tener estos datos embebidos dentro de otros no supondrá un gran problema a la hora de gestionar las actualizaciones.

La siguiente tabla muestra los datos de las colleciones con las que vamos a trabajar:

Colección	Información	Indices
Articles	▪ _id ▪ authors: Array con los nombres de los autores.	▪ _id
Incollections	▪ _id ▪ authors: Array con los nombres de los autores.	▪ _id
Inproceedings	▪ _id ▪ authors: Array con los nombres de los autores.	▪ _id
Authors	▪ _id ▪ authors: Array con los nombres de los autores.	▪ _id

1.2. Carga de los datos.

Los datos han sido cargados usando la herramienta **mongoimport**. Esta herramienta se ejecuta desde la línea de comandos y se instala junto con el propio MongoDB. Hay que tener en cuenta que, en caso de no existir la base de datos o la colección donde se indica que se deben cargar los datos, es el propio MongoDB el encargado de crearlas.

Para cargar nuestros datos se han ejecutado los siguientes comandos:

```
mongoimport --db=dblp --collection=articles articles/articles.json
mongoimport --db=dblp --collection=incollections incollections/incollections.json
mongoimport --db=dblp --collection=Inproceedings Inproceedings/Inproceedings.json
```

Al cargar los datos de esta forma, los *array* de autores y de ee han sido cargados como *arrays* de objetos, los cuales tienen un campo llamado **__VALUE** que contiene la cadena de texto. Es por ello que aplicamos un preprocesamiento para convertir estos campos en *arrays* formados por cadenas de texto. El código utilizado para ello:

```
db.articles.aggregate([{$addFields: {author: '$author._VALUE', title: '$title._VALUE',
  ee: '$ee._VALUE'}}, {$out: "articles"}])

db.incollections.aggregate([{$addFields: {author: '$author._VALUE', ee: '$ee._VALUE'}},
  {$out: "incollections"}])
```

En el caso de los *inproceedings*, también necesitamos cambiar el tipo del campo año ya que no ha sido reconocido como númerico al realizar la carga.

```
db.inproceedings.find().forEach(function(obj){
  db.inproceedings.update(
    {"_id": obj._id, 'year': {$exists : true}},
    {$set: {"year": NumberInt(obj.year)}
  })
})
```

2. Análisis

2.1. Listado de todas las publicaciones de un autor determinado.

Para esta query vamos a hacer uso de la potencia que nos ofrece el *aggregation framework*. Este framework funciona creando un pipeline en la que definiremos *stages*. La salida de un *stage* pasa a ser la entrada del *stage* siguiente. Con esto podemos realizar sobre los datos operaciones realmente potentes.

Para obtener el listado de las publicaciones, la query que debemos ejecutar es sencilla. En primer lugar filtramos por el nombre de autor que del cual queremos hacer la consulta, y a continuación concatenamos los *arrays* con los títulos de las publicaciones.

```
db.authors.aggregate([
  { $match : { _id : "Chin-Wang Tao" } },
  { $project: {publication: {$concatArrays:
    ["$incollections.title", "$articles.title", "$inproceedings.title"]}}
  }
] )
```

2.2. Número de publicaciones de un autor determinado.

Esta consula es similar a la anterior, pero debemos añadir un paso más en nuestro *pipeline* en el cual contaremos el número de puclicaciones.

```
db.authors.aggregate([
  { $match : { _id : "Chin-Wang Tao" } },
  { $project: {publication: {$concatArrays:
    ["$incollections.title", "$articles.title", "$inproceedings.title"]}}
  },
  {$project: {number_of_publications: {$size: "$publications"}}}
] )
```

2.3. Número de artículos en revistas para el año 2017.

```
db.articles.find({year: 2017}).count()
```

Aprovechando que esta query es simple, podemos ver la importancia que adquieren los índices en cuanto a tiempo de ejecución:

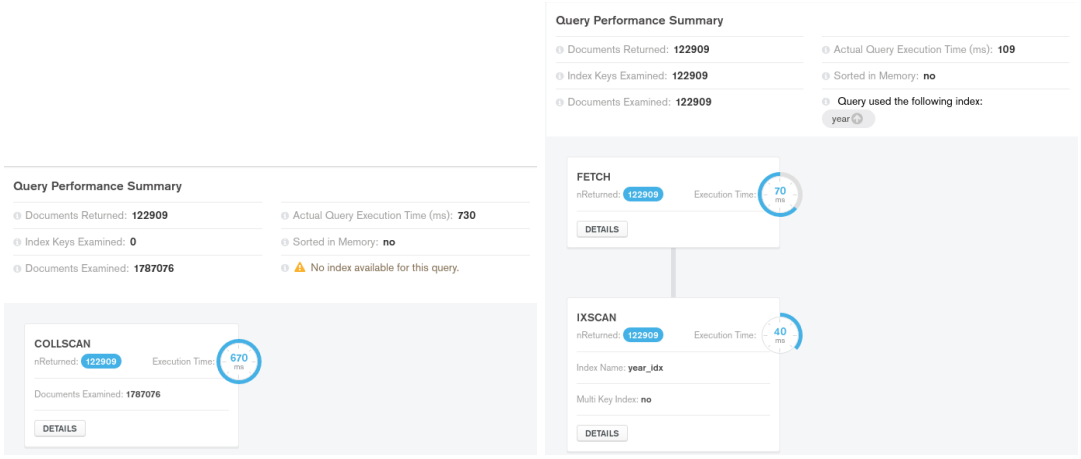

Figura 1: A la izquierda, la query sin indice, a la deracha, aplicando un índice sobre la columna year.

El tiempo de respuesta es casi 7 veces inferior al hacer uso del índice.

2.4. Número de autores ocasionales (menos de 5 publicaciones).

En esta ocasión no hará falta hacer uso del *aggregation framework*, si no que haremos una busqueda utilizando la función **find()**.

```
db.authors.find({$expr: {$lt: [{$concatArrays: ["$incollections", "$articles",
  "$inproceedings"]}, 5]}}).count()
```

2.5. Número de artículos y de artículos en congresos de los diez autores con más publicaciones en total.

Volvemos a usar *aggregation framework* y construimos un *pipeline* con varios pasos. Primero contamos el número de cada tipo de publicacion de cada uno de los autores. A coninuación, creamos un nuevo campo con el con **$addFields** con el total de publicaciones. Con **$sort** y **$limit** ordenamos por el campo recién creado y nos quedamos solo con los 10 primeros, es decir, con los autores que más publicaciones tienes. Por último, y con el *stage* **$project**, nos quedamos solo con los campos que nos interesan (el campo **_id** se incluye por defecto si no se especifica lo contrario).

```
db.authors.aggregate(
  [
    {$project: {
      "total_incollections": {$size: {$ifNull: ['$incollections', []]},},
      "total_inproceedings": {$size: {$ifNull: ['$inproceedings', []]},},
      "total_articles": {$size: {$ifNull: ['$articles', []]},}
    }},
    {$addFields: {
      "total_publications": { $add: [ "$total_incollections", "$total_inproceedings",
^^I^^I^^I^^I      "$total_articles" ] }
    }},
    {$sort : { total_publications : -1 } },
    {$limit : 10},
    {$project: {
      "total_articles": 1,
      "total_inproceedings": 1
    )}
  ],
  { allowDiskUse: true }
)
```

2.6. Número medio de autores por publicación.

En este caso vamos a tener que hacer consulta a varias coleciones y luego hacer calculos con los datos obtenidos en estas consultas.

En primer lugar realizamos consultas en cada una de las colecciones para obtener el número total de autores, incluyendo duplicados. Es decir, si por ejemplo, en un artículo han participado 3 autores, se sumara 3 a la cuenta de autores para esta colección. Estos resultados son sumados y el total lo dividimos por el número total de publicaciones guardadas.

```
(db.articles.aggregate([
  {$match: {author : {$exists: true}}},
  {$project: {num_authors: { $size: '$author' }}},
  {$group: {_id: '', num_authors: {$sum: '$num_authors'}}},
  {$project: {_id: 0, 'num_authors': '$num_authors'}}])
  .next()['num_authors'] +
db.incollections.aggregate([
```

```
  {$match: {author : {$exists: true}}},
  {$project: {num_authors: { $size: '$author' }}},
  {$group: {_id: '', num_authors: {$sum: '$num_authors'}}},
  {$project: {_id: 0, 'num_authors': '$num_authors'}}])
  .next()['num_authors'] +
db.inproceedings.find({author : {$exists: true}}).count())/
(db.articles.find().count() +
  db.incollections.find().count() +
  db.inproceedings.find().count())
```

2.7. Lista de coautores de un autor.

Para poder recuperar esta información, será necesario acceder a cada una de las publicaciones de las lista de publicaciones de un autor, "seguir" la referencia para obtener la publicación, y extraer los autores de esta. Para poder "seguir" una referencia a otro documento, *aggregation framework* cuenta con el *stage* **$lookup**, al que le decimos en que colección buscar y que campos tiene que utilizar a la hora de realizar la busqueda.

```
db.authors.aggregate([
  { $match : { _id : "Chin-Wang Tao" } },
  {$lookup:
    {
      from: "articles",
      localField: "articles._id",
      foreignField: "_id",
      as: "articles_detail"
    }
  },
  {$lookup:
    {
      from: "incollections",
      localField: "incollections._id",
      foreignField: "_id",
      as: "incollections_detail"
    }
  },
  {$project:{
      _id : 1,
      authors_detail : { $concatArrays: [ "$articles_detail", "$incollections_detail" ] }
    }
  },
  {$project: {
    "_id": 0,
    "results": {
      $reduce: {
        input: "$authors_detail.author",
        initialValue: [],
        in: { $concatArrays : ["$$value", "$$this"] }
      }
    }
  }
  },
  { $unwind : "$results" },
  {$group: {
    _id: "$results",
    }
```

```
  },
  {$group: {
      _id: null,
      coauthors: { $push: { _id: "$_id"} }
    }
  },
  {$project: {
      _id: 0,
      "coauthors": "$coauthors._id"
    }
  }
])
```

Conclusiones

Aún ninguna.

ANEXOS

A. Parseador de XML a JSON

En construcción.