Máster en Data Science. URJC
Esta es la asignatura

Este es el título.

AUTOR 1, AUTOR 2

6 de abril de 2018

Índice

Lista de Figuras

Glosario

BBDD Base de datos.

JSON JavaScript Object Notation.

XML eXtendible Markup Language.

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 B.

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 documentos almacenará 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"]}}
  }
] )
```

Como resultado obtenemos:

```
/* 1 */
{
    "_id" : "Chin-Wang Tao",
    "publication" : [
        "iPhone as Multi-CAM and Multi-viewer.",
        "On the robustness of stability of fuzzy control systems.",
        "Adaptive fuzzy PIMD controller for systems with uncertain deadzones.",
        "Design and analysis of region-wise linear fuzzy controllers.",
        "Adaptive Fuzzy Switched Swing-Up and Sliding Control for the Double-Pendulum-and-Cart
        ↪  System.",
        "An Approximation of Interval Type-2 Fuzzy Controllers Using Fuzzy Ratio Switching
        ↪  Type-1 Fuzzy Controllers.",
        "Adaptive fuzzy terminal sliding mode controller for linear systems with mismatched
        ↪  time-varying uncertainties.",
        "A reduction approach for fuzzy rule bases of fuzzy controllers.",
        "Fuzzy Sliding-Mode Formation Control for Multirobot Systems: Design and
        ↪  Implementation.",
        "Segmentation of Psoriasis Vulgaris Images Using Multiresolution-Based Orthogonal
        ↪  Subspace Techniques.",
        "Fuzzy control for linear plants with uncertain output backlashes.",
        "Iris Recognition Using Possibilistic Fuzzy Matching on Local Features.",
        "Radial Basis Function Networks With Linear Interval Regression Weights for Symbolic
        ↪  Interval Data.",
        "Adaptive fuzzy sliding mode controller for linear systems with mismatched
        ↪  time-varying uncertainties.",
        "Design of fuzzy controllers with adaptive rule insertion.",
        "Flexible complexity reduced PID-like fuzzy controllers.",
        "Interval competitive agglomeration clustering algorithm.",
        "Fuzzy sliding-mode control for ball and beam system with fuzzy ant colony
        ↪  optimization.",
        "Hybrid robust approach for TSK fuzzy modeling with outliers.",
        "Design of a Fuzzy Controller With Fuzzy Swing-Up and Parallel Distributed Pole
        ↪  Assignment Schemes for an Inverted Pendulum and Cart System.",
        "Nested design of fuzzy controllers with partial fuzzy rule base.",
        "Fuzzy hierarchical swing-up and sliding position controller for the inverted
        ↪  pendulum-cart system.",
```

```
"Design of a parallel distributed fuzzy LQR controller for the twin rotor multi-input
↪  multi-output system.",
"Fuzzy adaptive approach to fuzzy controllers with spacial model.",
"Simplified type-2 fuzzy sliding controller for wing rock system.",
"Unsupervised fuzzy clustering with multi-center clusters.",
"An advanced fuzzy controller.",
"A Novel Approach to Implement Takagi-Sugeno Fuzzy Models.",
"Fuzzy Swing-Up and Fuzzy Sliding-Mode Balance Control for a Planetary-Gear-Type
↪  Inverted Pendulum.",
"A Design of a DC-AC Inverter Using a Modified ZVS-PWM Auxiliary Commutation Pole and
↪  a DSP-Based PID-Like Fuzzy Control.",
"Robust fuzzy control for a plant with fuzzy linear model.",
"Comments on \"Reduction of fuzzy rule base via singular value decomposition\".",
"A Novel Fuzzy-Sliding and Fuzzy-Integral-Sliding Controller for the Twin-Rotor
↪  Multi-Input-Multi-Output System.",
"Robust L",
"A Fuzzy Logic Approach to Target Tracking",
"Index compression for vector quantisation using modified coding tree assignment
↪  scheme.",
"Hybrid SVMR-GPR for modeling of chaotic time series systems with noise and
↪  outliers.",
"Parallel Distributed Fuzzy Sliding Mode Control for Nonlinear Mismatched Uncertain
↪  Systems.",
"An approach for the robustness comparison between piecewise linear ",
"Interval fuzzy sliding-mode formation controller design.",
"Simplification of a fuzzy mechanism with rule combination.",
"Robust control of systems with fuzzy representation of uncertainties.",
"Checking identities is computationally intractable NP-hard and therefore human
↪  provers will always be needed.",
"A kernel-based core growing clustering method.",
"An Alternative Type Reduction Approach Based on Information Combination with Interval
↪  Operations for Interval-Valued Fuzzy Sliding Controllers.",
"Designs and analyses of various fuzzy controllers with region-wise linear PID
↪  subcontrollers.",
"Adaptive Bound Reduced-Form Genetic Algorithms for B-Spline Neural Network
↪  Training.",
"A New Neuro-Fuzzy Classifier with Application to On-Line Face Detection and
↪  Recognition.",
"Statistical and Dempster-Shafer Techniques in Testing Structural Integrity of
↪  Aerospace Structures.",
"Embedded support vector regression on Cerebellar Model Articulation Controller with
↪  Gaussian noise.",
"Vision-Based Obstacle Detection for Mobile Robot in Outdoor Environment.",
"Iris Recognition Using Gabor Filters and the Fractal Dimension.",
"A two-stage design of adaptive fuzzy controllers for time-delay systems with unknown
↪  models.",
"Errata: Iris recognition using Gabor filters optimized by the particle swarm
↪  algorithm.",
"Texture classification using a fuzzy texture spectrum and neural networks.",
"Iris recognition using Gabor filters optimized by the particle swarm algorithm.",
"FPGA implementation of improved ant colony optimization algorithm for path
↪  planning.",
"Medical image compression using principal component analysis.",
"Design of a Kinect Sensor Based Posture Recognition System.",
"A simplified interval type-2 fuzzy CMAC.",
```

```
        "Path-planning using the behavior cost and the path length with a multi-resolution
        ↪  scheme.",
        "Iris recognition using Gabor filters optimized by the particle swarm technique.",
        "Design of A Two-Stage Adaptive Fuzzy Controller.",
        "Face Detection Using Eigenface and Neural Network."
    ]
}
```

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" } },
  { $project: {publications: {$concatArrays:
    ["$incollections.title", "$articles.title", "$inproceedings.title"]}}
  },
  {$project: {number_of_publications: {$size: "$publications"}}}
] )
```

Como resultado obtenemos:

```
/* 1 */
{
    "_id" : "Chin-Wang Tao",
    "number_of_publications" : 64
}
```

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. Para ello, generamos el índice sobre el campo year de la siguiente forma:

```
db.articles.createIndex({year:1})
```

```
/* 1 */
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1.0
}
```

Vemos que el resultado es satisfactorio y que a partir de este momento, además del índice originario (sobre _id), tenemos índice sobre year.

La comparativa en tiempos de ejecución es la siguiente:

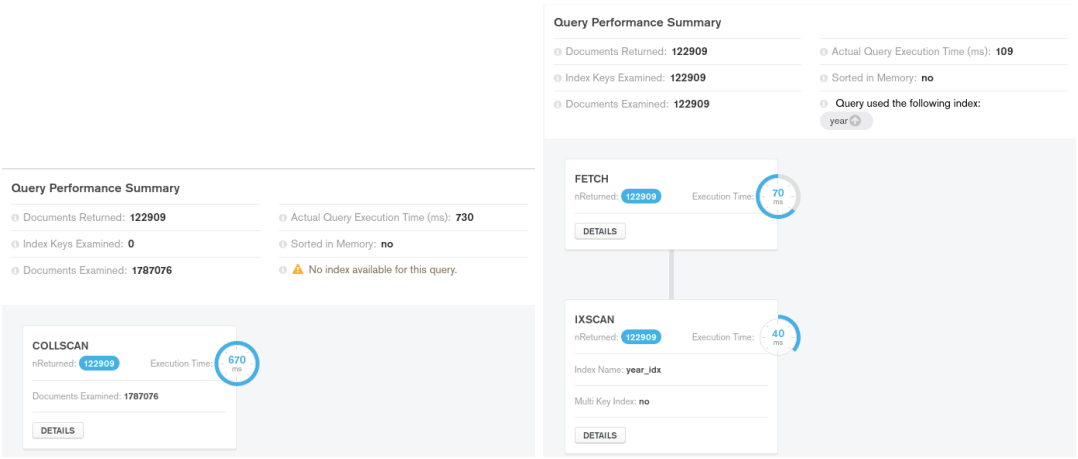

Figura 1: A la izquierda, la query sin índice, a la derecha, aplicando un índice sobre la columna year.

El tiempo de respuesta es casi 7 veces inferior al hacer uso del índice.
Como resultado, independientemente del uso de índice, obtenemos:

```
122909
```

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 búsqueda utilizando la función **find()**.

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

Como resultado obtenemos:

```
1306077
```

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 continuación, creamos un nuevo campo con el con **$addFields** con el total de publicaciones. Con **$sort** y **$limit** ordenamos por el campo recién creado y nos quedamos solo con los 10 primeros, es decir, con los autores que más publicaciones tienen. Por último, y con el *stage* **$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", "$total_inproceedings",
                  "$total_articles" ] }
```

```
    }},
    {$sort : { total_publications : -1 } },
    {$limit : 10},
    {$project: {
      "total_articles": 1,
      "total_inproceedings": 1
    }}
  ],
  { allowDiskUse: true }
)
```

Como resultado obtenemos:

```
/* 1 */
{
    "_id" : "H. Vincent Poor",
    "total_inproceedings" : 340,
    "total_articles" : 923
}

/* 2 */
{
    "_id" : "Lajos Hanzo",
    "total_inproceedings" : 371,
    "total_articles" : 607
}

/* 3 */
{
    "_id" : "Mohamed-Slim Alouini",
    "total_inproceedings" : 391,
    "total_articles" : 556
}

/* 4 */
{
    "_id" : "Philip S. Yu",
    "total_inproceedings" : 477,
    "total_articles" : 389
}

/* 5 */
{
    "_id" : "Wen Gao 0001",
    "total_inproceedings" : 525,
    "total_articles" : 335
}

/* 6 */
{
    "_id" : "Thomas S. Huang",
    "total_inproceedings" : 514,
    "total_articles" : 300
}

/* 7 */
```

```
{
    "_id" : "Witold Pedrycz",
    "total_inproceedings" : 135,
    "total_articles" : 659
}

/* 8 */
{
    "_id" : "Victor C. M. Leung",
    "total_inproceedings" : 355,
    "total_articles" : 432
}

/* 9 */
{
    "_id" : "Xiaodong Wang",
    "total_inproceedings" : 158,
    "total_articles" : 525
}

/* 10 */
{
    "_id" : "C.-C. Jay Kuo",
    "total_inproceedings" : 416,
    "total_articles" : 251
}
```

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())
```

Como resultado obtenemos:

```
1.647618039090564
```

2.7. Lista de coautores de un autor.

Para poder recuperar esta información, será necesario acceder a cada una de las publicaciones de la 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"
    }
  }
])
```

Como resultado obtenemos:

```
/* 1 */
{
    "coauthors" : [
        "Hung T. Nguyen 0002",
        "Seetharami R. Seelam",
        "Roberto A. Osegueda",
        "Wei-Yen Wang",
        "Vladik Kreinovich",
        "Cheng-Yuan Yang",
        "Sun-Yuan Kung",
        "Yao-Chu Hsueh",
        "Ana C. Holguin",
        "Yung-Chih Liu",
        "Chien-Ming Wang",
        "Meng-Cheng Yang",
        "Wiley E. Thompson",
        "Rustom Mamlook",
        "Y. C. Chen",
        "Chen-Chia Chuang",
        "Huei-Rong Chen",
        "Chunlin Chen",
        "Yeong-Hwa Chang",
        "Chen-Guan Chang",
        "Chia-Wen Chang",
        "Jin-Shiuh Taur",
        "J. H. Chang",
        "C. L. Tsai",
        "Mei-Lang Chan",
        "Heng-Yi Lin",
        "Chien-Chou Chen",
        "U. S. Chen",
        "Chin-Wang Tao",
        "Shun-Feng Su",
        "Chung-Chih Tsai",
        "Chia-Chu Hsu",
        "Chih-Ching Hsiao",
        "C. M. Wang",
        "Ching-Wen Yang",
        "Tsu-Tian Lee",
        "Gwo-Her Lee",
        "Hung-Wei Lin",
        "Jin-Tsong Jeng",
        "Tzuen Wuu Hsieh"
    ]
}
```

2.8. Edad de los 5 autores con el periodo de publicaciones más largo.

En primer lugar, y usando las facilidades de *aggregation framework*, agrupamos todas las publicaciones de un mismo autor en una lista única. A continuación nos quedamos de estas listas solo con el año de publicación de estas, y filtramos tanto el máximo como el mínimo de de estos años. Ya con esto, podemos calcular la diferencia entre ambos años y realizar una ordenación a partir de este campo para quedarnos con los cinco primeros.

```
db.authors.aggregate(
  [
    {$project: {
      publication: {$concatArrays: [
        {$ifNull: ['$incollections', []]},
        {$ifNull: ['$inproceedings', []]},
        {$ifNull: ['$articles', []]}]}
    }},
    {$addFields: {
      max_publication: { $max: "$publication.year"},
      min_publication: { $min: "$publication.year"}
    }},
    {$addFields: {
      age: { $subtract: [ "$max_publication", "$min_publication"] }
    }},
    {$sort : { age : -1 } },
    {$limit : 5},
    {$project: {
      _id: 1,
      age: 1
    }}
  ],
  { allowDiskUse: true }
)
```

Como resultado obtenemos:

```
/* 1 */
{
    "_id" : "Alan M. Turing",
    "age" : 75
}

/* 2 */
{
    "_id" : "Rudolf Carnap",
    "age" : 71
}

/* 3 */
{
    "_id" : "David Nelson",
    "age" : 68
}

/* 4 */
{
    "_id" : "Eric Weiss",
    "age" : 64
```

```
}

/* 5 */
{
    "_id" : "Bernard Widrow",
    "age" : 64
}
```

2.9. Número de autores novatos.

Caso idéntico al anterior, cambiando las fases finales para quedarnos con aquellos cuya "edad" es inferior a 5 años, y contar el número de elementos que obtenemos.

```
db.authors.aggregate(
  [
    {$project: {
      publication: {$concatArrays: [
        {$ifNull: ['$incollections', []]},
        {$ifNull: ['$inproceedings', []]},
        {$ifNull: ['$articles', []]}]}
    }},
    {$addFields: {
      max_publication: { $max: "$publication.year"},
      min_publication: { $min: "$publication.year"}
    }},
    {$addFields: {
      "age": { $subtract: [ "$max_publication", "$min_publication"] }
    }},
    {'$match':{'age': {'$lt': 5}}}
  ],
  { allowDiskUse: true }
).itcount()
```

Como resultado obtenemos:

```
999264
```

2.10. Porcentaje de publicaciones en revistas con respecto al total de publicaciones.

Al igual que hicimos cuando calculamos el número medio de autores, que tuvimos que hacer varias queries a cada una de las colecciones, para este caso vamos a volver a operar con diversas queries aplicadas sobre las distintas colecciones. Necesitaremos obtener el número de documentos en cada colección y con estos datos podemos calcular el porcentaje de publicaciones en revistas como número de estas dividido entre el número total de publicaciones.

```
db.articles.find().count()*100/(
  db.articles.find().count() +
  db.incollections.find().count() +
  db.inproceedings.find().count())
```

Como resultado obtenemos:

```
49.15410131860515
```

Conclusiones

Aún ninguna.

ANEXOS

A. Creación de vistas

De forma similar que en bases de datos relaciones, es posible crear vistas. Las vistas son consultas sobre diferentes tablas a través de los campos que designemos. Una vez creada una vista, las consultas se realizan de la misma forma que si fuese una colección, pudiendo filtrar por alguno de sus campos.

En nuestro caso, hemos creado la vista *publications_extended*. Esta vista, a partir de la colección *authors* que contiene un documento por cada autor y algunos campos básicos de cada tipo de documento, incluyendo el campo _id, cruzando con el resto de colecciones con diferentes publicaciones, conseguimos unificar toda la información extendida sobre una vista. Al incluir nuevos documentos en las colecciones de origen, la vista automáticamente devuelve dichos registros. La definición es la siguiente:

```
db.createView (
<<<<<<< HEAD
  "publications_extended",
  "authors",
  [
    { $lookup: {
      from: "articles", localField: "articles._id",
^^IforeignField: "_id", as: "articles_extended" } },
    { $lookup: {
      from: "incollections", localField: "incollections._id",
^^IforeignField: "_id", as: "incollections_extended" } },
    { $lookup: {
      from: "inproceedings", localField: "inproceedings._id",
^^IforeignField: "_id", as: "inproceedings_extended" } },
    { $project: {
      articles_extended: 1, incollections_extended: 1,
^^Iinproceedings_extended: 1}}
  ]
)
```

B. Parseador de XML a JSON

Para realizar el parseo de los datos hemos decidido hacer uso de PySpark para aprovechar la gestión de la memoria que hace. ¿Por qué de esta decisión?. Intentar procesar el fichero eXtendible Markup Language (XML) directamente con un script de Python provocaba que las máquinas donde se ejecuba acabaran bloqueandose debido a que se llegaba al límite de la capacidad de la memoria, y por lo que se ha podido comprobar, los paquetes encargados de este parseo no gestionan correctamente estos escenarios.

Otra alternativa era dividir el fichero XML en bloques y procesarlos de manera individual. Así se reduce la carga de datos en memoria y se consigue no llegar al límite de la máquina. Esta técnica puede se más compleja ya que requiere leer el fichero línea a línea y procesarla para saber si podemos "cortar" sobre ella no separar un elemento valido (un *articles* en dos bloques.

Debido a la facilidad de uso que nos ofrece PySpark, y aprovechando que es conetenido de otra materia de este master, lo hemos visto como la opción más recomendable.

B.1. Pasos

Para leer el XML abrimos una shell de pyspark con el paquete de java spark-xml:

```
pyspark --packages com.databricks:spark-xml_2.11:0.4.1
```

Desde pyspark ejutamos para cada tipo de publicación el siguiente mandato:

```
df = spark.read.format('com.databricks.spark.xml').option("rowTag",
  "incollection").load('./dblp.xml')
```

Al no tener un esquema definido, es el propio paquete de **Spark** el que intenta inferirlo por si mismo. Esto causa problemas al intentar parsear algunos campos. Por ello lo mejor es definir nuestro propio esquema definiendo todos los campos como si se tratasen de "Strings". Ya que solo utilizamos este dataframe para transformar el *XML* a *JSON*, y no vamos a realizar ningún tipo de validación u operación con el **Datafame** que se crea, tratar todos los campos como "Strings" es más que suficiente.

```
from pyspark.sql.types import StructField, StringType, StructType

custom_types = []
for c in df.columns[1:]:
  custom_types.append(StructField(str(c), StringType(), nullable=True))

custom_schema = StructType(custom_types)
```

Una vez que tengamos creado el **DataFrame**, solo nos quedará volcarlo como fichero, o más bien, como múltiples ficheros, ya que por el mecanismo interno de funcionamiento de Spark, la salida sera escrita en múltiples ficheros. Tendremos por tanto que indicarla a la función *json()* el directorio donde queremos que vuelque estos ficheros.

```
df.write.json('./incollection')
```

Una vez terminado, nos encontrameros que dentro del directorio anterior, se han creado varios ficheros cuyos nombres tiene el formato *PARTXXXXX_.json*. Para unirlo todos en uno solo (en caso de querer hacerlo), podemos usar las **tools** básicas que nos ofrece el ofrece el sistema operativo, en nuestro caso, una distribución basada en **Debian**:

```
cat PART* > out.json
```