Sistemas de Gestión de Datos y de la Información
Máster en Ingeniería Informática, 2018/19
Práctica 3

Fecha de entrega: miércoles 5 de diciembre de 2018, 10:00h

Objetivos mínimos
Realizar consultas básicas en MongoDB desde Python utilizando pymongo.

Entrega de la práctica
La práctica se entregará en un único fichero **GrupoXX.zip** mediante el Campus Virtual de la asignatura. Este fichero **ZIP** contendrá dos ficheros: `consultas.py` con las funciones Python necesarias para realizar las consultas y modificaciones que se detallan en el enunciado, y `consultas.js` con las funciones JavaScript necesarias para realizar las consultas avanzadas utilizando el *aggregation framework* y *MapReduce*.

Lenguaje de programación
Python **3.6 o superior**

Ficheros
`usuarios.json` y `peliculas.json`.

Calificación
Cada apartado tiene un peso del 50 %. Se valorará la corrección, la claridad y organización del código, además de la eficiencia de las consultas.

Declaración de autoría e integridad
Todos los ficheros entregados contendrán una cabecera en la que se indique la asignatura, la práctica, el grupo y los autores. Esta cabecera también contendrá la siguiente declaración de integridad:

> *NombreCompletoAlumno1* y *NombreCompletoAlumno2* declaramos que esta solución es fruto exclusivamente de nuestro trabajo personal. No hemos sido ayudados por ninguna otra persona ni hemos obtenido la solución de fuentes externas, y tampoco hemos compartido nuestra solución con nadie. Declaramos además que no hemos realizado de manera deshonesta ninguna otra actividad que pueda mejorar nuestros resultados ni perjudicar los resultados de los demás.

No se corregirá ningún fichero que no venga acompañado de dicha cabecera.

En esta práctica consideraremos una base de datos MongoDB `sgdi_pr3` que almacena información sobre **usuarios** y **películas**. Para ello dispone de dos colecciones `usuarios` y `peliculas` con el siguiente aspecto (todos los documentos tienen el mismo número y tipo de campos):

```
COLECCION USUARIOS
{
  "_id" :"fernandonoguera",
  "nombre" :"Pedro",
  "apellido1" :"Cordero",
  "apellido2" :"Bustos",
  "sexo" :"M",
  "gustos" :[ "terror", "comedia", "tragedia" ],
  "email" :"hporcel@arregui-belmonte.com",
  "edad" :54,
  "password" :"c46526e4f34352111b9f98feaacf1338b59d8d15",
  "direccion" :{ "cod_postal" :"73182",
                 "numero" :90,
                 "puerta" :"C",
                 "pais" :"Alemania",
                 "piso" :3,
                 "nombre_via" :"Camino de Laura Rebollo",
                 "tipo_via" :"Avenida" }
  "visualizaciones" :[
    { "_id" :ObjectId("583ef650323e9572e2813189"),
      "titulo" :"Ex delectus vel dicta delectus.",
      "fecha" :"1995-04-06" },
    { "_id" :ObjectId("583ef651323e9572e2813dd5"),
      "titulo" :"Ipsam repudiandae dolorem libero voluptatibus.",
      "fecha" :"1978-12-27" }
  ],
}
```

```
COLECCION PELICULAS
{
  "_id" :ObjectId("583ef650323e9572e2813189"),
  "director" :"Lourdes Sacristan Bernal",
  "titulo" :"Ex delectus vel dicta delectus.",
  "lanzamiento" :1967,
  "pais" :["Myanmar", "Azerbaiyan" ]
}
```

Podéis cargar datos de prueba para estas dos colecciones con el binario `mongoimport` a partir de los ficheros `usuarios.json` y `peliculas.json`.

1. MongoDB en Python [Objetivo mínimo, 50 %]

Completar el esqueleto `consultas.py` para realizar las operaciones de consulta que se presentan a continuación. Cada consulta está encapsulada en una función Python que acepta como parámetros los valores necesarios para realizar la consulta y se comunica con MongoDB utilizando la librería pymongo.

El resultado de estas funciones debe ser el **objeto `pymongo.cursor.Cursor` con los resultados de la consulta**.

Las consultas que se deben implementar son:

1. (6.5 %) Obtener la **fecha** y **título** de las primeras `n` películas (según el orden en el que aparecen en el documento, no su fecha) visualizadas por el usuario `user_id`. Ej:
   `usuario_peliculas ( 'fernandonoguera', 3 )`

2. (6 %) Devolver el **_id**, **nombre** y **apellidos** de los primeros `n` usuarios a los que les gusten una serie de tipos de película (todos a la vez). Ej:
   `usuarios_gustos ( ['terror', 'comedia'], 5 )`

3. (6.5 %) Obtener el **_id** de los usuarios de un determinado `sexo` y con una edad comprendida entre `edad_min` y `edad_max` (incluidas). Ej:
   `usuario_sexo_edad ('M', 50, 80)`

4. (8 %) Devolver el **nombre** y **apellidos** de los usuarios cuyo primer y segundo apellido coinciden, ordenados por edad de manera ascendente. Ej:
   `usuarios_apellidos ()`

5. (6.5 %) Recuperar el **título** de las películas cuyo director tenga un nombre que empiece por un determinado `prefijo`. Ej:
   `pelicula_prefijo ( 'Yol' )`

6. (6.5 %) Obtener el **_id** de aquellos usuarios que tienen exactamente `n` gustos, ordenados por edad descendente. Ej:
   `usuarios_gustos_numero( 6 )`

7. (10 %) Devolver el **_id** de los usuarios que vieron la película `id_pelicula` entre dos fechas `inicio` y `fin`. Ej:
   `usuarios_vieron_pelicula ( '583ef650323e9572e2812680', '2015−01−01', '2016−12−31')`

2. *Aggregation Pipeline* y MapReduce [50 %]

Además de las consultas usuales, MongoDB admite dos mecanismos para realizar consultas complejas. El primero de ellos es el *aggregation pipeline* (`https://docs.mongodb.org/manual/core/aggregation-pipeline/`), que permite aplicar distintas etapas de manera ordenada sobre una colección. Por otro lado MongoDB soporta *MapReduce* (`https://docs.mongodb.org/manual/core/map-reduce/`), con un comportamiento similar al que hemos explicado durante el curso (*¡pero revisad bien la documentación que hay sorpresas!*).

En este apartado implementaremos consultas *avanzadas* utilizando estos mecanismos. Todas las consultas se realizarán sobre las mismas colecciones del apartado anterior, dentro de la base de datos `sgdi_pr3`. Por simplicidad vamos a dejar de lado Python y centrarnos en cómo realizar las consultas desde el *shell* de Mongo mediante funciones JavaScript. Por tanto en este apartado se entregará un único fichero `consultas.js` cuyo esqueleto se puede descargar del Campus Virtual. Cada consulta está encapsulada en una función (`agg1()`–`agg4()` para *aggregation pipelines* y `mr1()`–`mr4()` para *MapReduce*) que **debe devolver los resultados de la consulta**, es decir, no debe almacenarlos en ninguna colección. Podéis declarar las funciones auxiliares que consideréis necesarias.

Consultas con *Aggregation Pipeline*

1. (6.25 %) Listado de *país-número de películas* rodadas en él, ordenado por número de películas descendente y en caso de empate por nombre país ascendente.

2. (6.25 %) Listado de los 3 tipos de película más populares entre los usuarios de los `'Emiratos Árabes Unidos'`, ordenado de mayor a menor número de usuarios que les gusta. En caso de empate a número de usuarios, se usa el tipo de película de manera ascendente.

3. (6.25 %) Listado de *país-*(*edad mínima, edad-máxima, edad media)* teniendo en cuenta únicamente los usuarios mayores de edad, es decir, con más de 17 años. Los países con *menos de 3 usuarios mayores de edad* **no deben aparecer en el resultado**.

4. (6.25 %) Listado de *título película-número de visualizaciones* de las 10 películas más vistas, ordenado por número descencente de visualizaciones. En caso de empate, romper por título de película ascendente.

Consultas con MapReduce

1. (6 %) Listado de *país-número de películas* rodadas en él.

2. (6 %) Listado de *rango de edad-número de usuarios*. Los rangos de edad son periodos de 10 años: $[0, 10)$, $[10, 20)$, $[20, 30)$, etc. Si no hay ningun usuario con edad en un rango concreto dicho rango no debería aparecer en la salida.

3. (7 %) Listado de *país-*(*edad mínima, edad-máxima, edad media)* teniendo en cuenta únicamente los usuarios con más de 17 años.

4. (6 %) Listado de *año-número de visualizaciones veraniegas*, donde una «visualización veraniega» es aquella que se ha producido en los meses de junio, julio o agosto.