DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico #2

5 de Noviembre de 2016 — Base de Datos

Integrante	LU	Correo electrónico
Pedro Rodriguez	197/12	pedro3110.jim@gmail.com
Lucas Tavolaro Ortiz	322/12	tavo92@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - Pabellón I
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Argentina
Tel/Fax: (54 11) 4576-3359
http://exactas.uba.ar

Índice

1. Introducción

En el presente trabajo llevamos nuestro proyecto a MongoDB, analizaremos maneras de representarlo para distintas queries con sus ventajas y desventajas y escribiremos el código de cada map-reduce.

2. Documentos para Mongo

A continuación vamos a definir el documento para Mongo en cada una de las queries dadas y argumentar la elección de dicho documento contando, a alto nivel, como serían los inserts, mantenimiento y busqueda con map-reduce.

2.1. Crímenes cometidos por persona

1. Para mayor nro. de crimenes cometidos por alguna persona
2. Para nro de crimenes promedio cometidos por criminales

Analizaremos la querie que nos pide el mayor número de crimines cometidos por alguna persona y número de crímenes promedio cometidos. Elegimos diseñar el documento expuesto a continuación:

```
[
        {
                dniCulpable: 37206752,
                idCaso: 2
        }
]
```

Guardaremos en cada documento el **dniCulpable** de la persona que fue encontrada culpable del crime **idCaso**.

2.1.1. Insert

Cada vez que una persona es hallada culpable de un nuevo caso, creamos un nuevo documento con su dni y el caso en cuestión. Esto es una operación muy rápida pues no depende de otros valores.

2.1.2. Mantenimiento

Las ventajas de esta manera de representar la información están en su mantenimiento. Cada documento ocupa poco espacio y, al no depender de nada más, es facil de mantener y escalar.

En un principio analizamos la posibilidad de tener un arreglo de casos por cada dni, pero esto requería mantener una estructura que no sabemos cuanto puede crecer en el tiempo y puede generar problemas al alcanzar el limite de disco. Por eso decidimos utilizar algo más sencillo, pero que podamos garantizar facilmente su mantenimiento y escalabilidad.

2.1.3. Map-Reduce

El map devuelve para una única clave, todos los crímenes cometidos por cada dni. Se recorre el array de tuplas, se arma un mapa que guarda para cada dni la cantidad de crímenes cometidos por este, y posteriormente se recorre dicho mapa y devuelve el máximo valor en un caso, y el promedio en otro. En el ejemplo, ponemos el caso en que calcula el máximo.

```
db.doc1.mapReduce(
    function() {
        emit(1, (this["dniCulpable"], 1) );
    },
    function(key,value) {
        var mapa = {};
        for(i = 0; i < value.length; i++) {
            mapa[value[i][0]] = mapa[value[i][0]] + 1;
        }
```

```
        var res = 0;
        for(var elem in map) {
                res = max(res, elem.value);
        }
        return res;
)
```

2.2. Ciudades con mayor número de crímenes

Analizaremos la querie que nos pide las 10 ciudades con mayor número de crímenes. Elegimos el documento expuesto a continuación:

```
1       [
2               {
3                       idCiudad: 4,
4                       dniCulpable: 37206752
5               }
6       ]
```

Guardaremos en cada documento, una referencia a la ciudad en **idCiudad** y un culpable de algún crimen como **dniCulpable**.

2.2.1. Insert

Cada vez que se reporte un nuevo crimen, creamos un documento nuevo con la información.

2.2.2. Mantenimiento

Es similar a la anterior querie analizada.

2.2.3. Map-Reduce

El map nos debe calcular para ciudad, cuantos crímenes tiene (idea similar a la querie anterior). El reduce, debe poder ordenarnos la ciudad según sus crimenes y quedarnos con las 10 mas grandes. Para hacer esto, creamos un heap de tamaño 10, y al recorrer el mapa, insertamos de forma ordenada cada valor.

```
db.doc2.mapReduce(
    function() {
        emit(1, (this["idCiudad"], 1) );
    },
    function(key,value) {
        var mapa = {};
        for(i = 0; i < value.length; i++) {
            mapa[value[i][0]] = mapa[value[i][0]] + 1;
        }
        var res = binaryHeap(10);
        for(var elem in map) {
                res.insert(elem.value);
        }
        return res;
)
```

2.3. Personas como testigos en casos

La querie consiste en responder cuántas personas se han visto involucradas como testigos en el mayor número de casos. Construimos el siguiente documento:

```
[
        {
                dniTestigo: 27893435,
                idCasosInvolucradaComoTestigo:
                        [
                                {idCaso: 1},
                                {idCaso: 4},
                                {idCaso: 6}
                        ]
        }
]
```

En **dniTestigo** representamos de que testigo se trata y luego tenemos un arreglo denominado **idCasosInvolucradaComoTestigo** en donde cada elemento es un caso en donde esa persona es testigo.

Si bien podiamos optar por un diseño similar a los anteriores, decidimos probar con otro que tiene otras ventajas y desventajas que marcaremos.

2.3.1. Insert

El Insert es mas costoso que en los anteriores pues debemos buscar el documento que contenga el dni del testigo del nuevo caso. Notemos que en caso de no existir un documento debemos crear uno nuevo. Una vez que lo tenemos, debemos agregar un nuevo caso al arreglo.

2.3.2. Mantenimiento

Mencionamos este caso anteriormente. La desventaja que tiene es que va a ir creciendo de tamaño cada documento y eventualmente puede llenar el disco. La solución que se nos ocurre es que, en caso de pasar eso, creamos un nuevo documento para ese testigo en donde a partir de ese momento iremos agregando los casos. En el map-reduce asumiremos que tenemos este diseño ya que es más interesante.

2.3.3. Map-Reduce

El map debe decirnos, para un testigo, en cuantos casos estuvo. Esto es simplemente la suma del largo del arreglo **idCasosInvolucradaComoTestigo** en cada documento en el que aparezca el dni. El reduce debemos primero determinar cuál es que apareció en más casos y luego retornar todas las personas que hayan aparecido en exactamente esa cantidad de casos que calculamos.

Este Map-Reduce es más eficiente que con el diseño usado anteriormente pues recorre muchisimos menos documentos. Recordemos que antes existía un documento por caso, y acá cada documento va a seguir creciendo hasta que no quede más lugar en el disco.

```
db.doc3.mapReduce(
    function() {
        emit(this["dniTestigo"], this["idCasosInvolucradaComoTestigo"].length);
    },
    function(key,value) {
        maximo = value[0];
        for(i = 1; i < value.length; i++) {
            if (maximo < value[i]) {
                maximo = value[i];
            }
        }
        var res = 0;
        for(i = 0; i < value.length; i++) {
                if (value[i] == maximo) {
                        res = res + 1;
                }
```

```
        }
        return res;
)
```

2.4. Casos con mayor número de involucrados o evidencias

Para esas 2 queries utilizaremos un mismo diseño que es el siguiente:

```
[
        {
                idCaso: 4,
                {
                        idEvidencias:
                                [
                                        {idEvidencia: 3},
                                        {idEvidencia: 4}
                                ],
                        dniInvolucrados:
                                [
                                        {dniInvolucrado: 37206752}
                                ]
                }
        }
]
```

En donde usaremos **idCaso** como clave, y para cada uno tendremos las distintas evidencias en **idEvidencias** y los involucrados en **dniInvolucrados**.

2.4.1. Insert

A lo largo del tiempo, irán apareciendo evidencias nuevas y gente involucrada. Como en la anterior querie, debemos actualizar el documento agregando dicha información según que caso se trate.

2.4.2. Mantenimiento

A diferencia del anterior, en este caso podemos pensar que es realmente muy dificil que llenemos un disco con este documento pues el orden de magnitud de las evidencias y los involucrados no parece ser tanto. En caso de llevar acabo un sistema así, siempre se puede consultar un limite para poder tomar una decisión de diseño de estas características. Además de esta diferencia, en caso de llenarse el disco podemos utilizar una técnica similar a la ya expuesta en la querie anterior.

2.4.3. Map-Reduce

Dependiendo la querie, dado un caso deberemos contar evidencias o involucrados. En el próximo ejemplo, sólo vamos a contar evidencias. En el map, mapeamos a un mismo key tuplas con el idCaso y la cantidad de evidencias correspondientes a dicho caso. En el reduce, recorremos el arreglo de valores **value**, y vamos guardando en un mapa para cada idCaso, la cantidad de evidencias que le corresponden. En la variable *maximo*, vamos actualizando en cada paso la máxima cantidad de evidencias que tiene algún caso. Luego, recorremos el mapa y devolvemos todos los idCaso cuyo valor es igual a la variable *maximo*.

```
db.doc4.mapReduce(
    function() {
        emit(1, (this["idCaso"], this["idEvidencias"].length) );
    },
    function(key,value) {
        var maximo = 0;
        var mapa = {};
        for(i = 0; i < value.length; i++) {
```

```
            mapa[value[i][0]] = mapa[value[i][0]] + value[i][1];
            maximo = max(maximo, mapa[value[i][0]]);
        }
        var res = new Array();
        for(var elem in mapa) {
                if (elem.value == maximo) {
                        res.push(elem.key);
                }
        }
        return res;
)
```

2.5. Crimenes cometidos en los últimos 45 días

La última querie nos pide la cantidad de crímenes de cada tipo cometidos en los ultimos 45 dias. Elegimos el siguiente diseño para el documento:

```
1   [
2       {
3           idTipoCrimen: 4,
4           cantCrimenesPorFecha:
5               {
6                   '2014-10-08': 4,
7                   '2015-03-10': 1,
8                   '2016-10-10': 2
9               }
10      }
11  ]
```

En donde **idTipoCrimen** es la clave que representa un tipo de crimen, y el mapa **cantCrimenesPorFecha**, cuyas claves son fechas y los valores cantidad de crímenes en cada una. Este documento tendrá la propiedad de que solo mantendremos fechas en los últimos 45 días guardadas (En Mantenimiento desarrollaremos mejor este punto).

2.5.1. Insert

Buscamos el tipo de crimen correspondiente para tener el documento (o crear un nuevo si no existe). Luego, buscamos en el diccinario la clave correspondiente a la fecha, e incrementamos su valor. En caso de no existir dicha clave, la creamos con el valor 1.

2.5.2. Mantenimiento

Para optimizar el uso del disco, creamos un documento para cada tipo de crimen, y dicho documento tiene una cantidad de espacio acotado pues puede contener como mucho 45 fechas distintas. Esto es una ventaja a la hora de escalar esta estructura.

Para poder garantizar la propiedad de que solo contenga crimenes de los últimos 45 días, cuando cambia de día debemos realizar una tarea de mantenimiento. Hay que recorrer el diccionario correspondiente a cada tipo de crimen, y borrar las entradas correspondientes a fechas de hace más de 45 días. Este proceso puede ser rápido, pues no creemos que existan muchisimos tipos de crímenes y como mencionamos, hay como mucho 45 fechas. Esto quiere decir que en este proceso eliminaremos como mucho 1 de ellas, ya que al cambia de día alguna puede haber pasado a ser 46 días antes.

2.5.3. Map-Reduce

En el map tendremos el tipo de crimen como la clave, y como valor emitiremos las cantidades de crímenes registradas en cada fecha. En el reduce calcularemos la suma de todos los crímenes en las fechas que hay guardadas (recordemos que todas están incluidas en los últimos 45 días).

```
db.doc5.mapReduce(
    function() {
        for (i = 0; i < this["cantCrimenesPorFecha"].length; i++) {
                for (var elem in this["cantCrimenesPorFecha"][i]) {
                                emit(this["idTipoCrimen"], elem.value);
                        }
        }
    },
    function(key,value) {
        var res = 0;
        for (i = 0; i < value.length; i++) {
                res = res + value[i];
        }
        return res;
    }
)
```

3. Conclusiones

En el trabajo realizamos distintos tipos de documentos en donde pudimos ver que cada uno tiene ciertas ventajas y desventajas. Nos pareció enriquecedor podes llevar adelante las ideas que tuvimos para poder realizar un desarrollo de como funciona cada una. A modo de cierre, podemos concluir que el diseño del documento es algo que necesita pensarse bien ya que define mucho de todo lo demás. En particular es importante poder saber, en casos reales, como se va a comportar el documento, es decir, cada cuanto tenemos inserts, que recursos tenemos, si podemos realizar operaciones de mantenimiento frecuentemente, que queries y con que frecuencia recibiremos, etc. Con esta información clara, se puede tomar una buena decisión en cuanto al diseño para poder elegir la que tenga las ventajas que necesita nuestro problema.