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 búsqueda con map-reduce.

2.1. Documento 1 y consultas asociadas

Elegimos diseñar el documento **doc1**. A continuación exponemos un ejemplo:

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

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

El JSONSchema correspondiente sería:

```
{
        "type" : object,
        "properties": {
                "dniCulpable" : {"type" : integer},
                "idCaso" : {"type" : integer}
        }
}
```

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 límite de disco. Por eso decidimos utilizar algo más sencillo, pero que podamos garantizar facilmente su mantenimiento y escalabilidad.

2.1.3. Map-Reduce para mayor número de crimenes cometidos por alguna persona

Para resolver la consulta empleamos dos map-reduce.

En el primero, map11 mapea a cada criminal, un array de longitud igual a la cantidad de crímenes que este cometió. Luego, el reduce11 realiza una agregación, calculando cuántos crímenes cometió cada criminal.

En el segundo, map12 mapea a una misma clave todos los pares $< criminal, cantidadCrimenes >$. Y el reduce 12, se ocupa de buscar cuál fue la máxima cantidad de crímenes cometida por algún criminal, y devolver dicho valor.

```
map11 = function () {
    emit( this . dniCulpable , 1 );
};
var reduce11 = function(key,values) {
    return Array.sum( values );
};
var map12 = function () {
    emit( 1, this.value );
};
var reduce12 = function(key, values) {
    max = values[0];
    for (var i=0; i<values.length; i++) {
        if(values[i] > max) {
            max = values[i];
        }
    }
    return max;
};
db . doc1 . mapReduce (map11, reduce11, {out: "temp1"} );
db . temp1 . mapReduce (map12, reduce12, {out: "ej1"});
```

2.1.4. Map-Reduce para el número de crímenes promedio cometidos por criminales

Para resolver esta segunda consulta, también empleamos dos map-reduce.

Para el primer map-reduce, utilizamos el mismo que utilizamos en el ejercicio anterior: usamos la función map11 y reduce11, para obtener para cada criminal, la cantidad de crímenes que cometió.

Para el segundo map-reduce, utilizamos las misma función de mapeo map12 del ejercicio anterior, para mapear a una misma clave todas las cantidades de crímenes cometidas por los distintos criminales. Luego, usamos la función reduce21 para calcular el promedio de estos valores.

```
var reduce21 = function(key, values) {
    sum = 0;
    cnt = 0;
    for (var i=0; i<values.length; i++) {
        cnt = cnt + 1;
        sum = sum + values[i];
    }
    return sum / cnt;
};
db . doc1 . mapReduce (map11, reduce11, {out: "temp2"} );
db . temp2 . mapReduce (map12, reduce21, {out: "ej2"} );
```

2.2. Documento 2 y consultas asociadas

Elegimos el documento **doc2**. A continuación, un ejemplo:

```
        [
                {
                        idCiudad: 4,
                        idCrimen: 12423,
                        anio: 2001
                }
        ]
```

Guardaremos en cada documento, una referencia a la ciudad en **idCiudad**, un crimen como **idCrimen** y el **anio** en que dicho crimen fue cometido.

El JSONSchema correspondiente sería:

```
{
        "type" : object,
        "properties" : {
                "idCiudad" : {"type" : integer},
                "idCrimen" : {"type" : integer},
                "anio" : {"type" : integer}
        }
}
```

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 para las 10 ciudades con mayor cantidad de crímenes

Para resolver esta consulta, utilizamos la función map31, que devuelve para ciudad, un vector con un 1 por cada crimen cometido en dicha ciudad. Luego, la función reduce31 se ocupa de calcular la suma total de este vector para cada ciudad. Para obtener sólo las 10 ciudades con mayor cantidad de crímenes, debemos ordenar el documento devuelto por este map-reduce, y quedarnos sólo con las primeras 10 entradas.

```
var map31 = function() {
    emit(this . idCiudad, 1);
};
var reduce31 = function(key, values) {
    return Array.sum(values);
};
db . doc2 . mapReduce(map31, reduce31, {out:"temp3"} );
db.temp3.find().sort({value: -1}).limit(10);
```

2.2.4. Map-Reduce para cantidad de crímenes por localidad y año

En este caso, resolvemos la consulta con un único map-reduce. En el map71, lo que hacemos es recorrer todo el documento, y mapear cada crimen según la clave doble $< ciudad, anio >$. Esta función de mapeo va a devolver un arreglo con un 1 por cada crimen cometido en una tupla $< ciudad, anio >$. Luego, el reduce71 lo único que debe hacer es sumar los valores de este arreglo, para obtener la cantidad de crímenes obtenidos en cada par de localidad y año posibles.

```
var map71 = function() {
    emit(
    {
        ciudad: this . idCiudad,
        anio: this . anio
    }, 1 );
};
var reduce71 = function(key, values) {
    return Array.sum(values);
};
db . doc2 . mapReduce(map71, reduce71, {out:"ej7"} );
```

2.3. Documento 3 y consultas asociadas

Construimos el documento **doc3**.

Para este punto, decidimos probar con otro modelo de documento, que tiene otras ventajas y desventajas respecto de los utilizados anteriormente, que marcaremos más adelante.

Una instancia de ejemplo para este documento podría ser:

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

Guardaremos en cada documento, una referencia al testigo con **dniTestigo**, y para cada uno de estos, un array **idCasosInvolucradaComoTestigo**, que tiene en cada entrada un **idCaso** con los casos en los que el testigo estuvo involucrado.

El JSONSchema correspondiente sería:

```
{
        "type" : object,
        "properties": {
                "dniTestigo" : {"type" : integer},
                "idCasosInvolucradaComoTestigo" : {
                        "type" : array,
                        "casos": {
                                "type" : object,
                                "properties": {
                                        "idCaso" : {"type" : integer}
                                }
                        }
                }
        }
}
```

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 para personas que fueron testigos en la mayor cantidad de casos

Para resolver esta consulta, empleamos dos map-reduce.

En el primero, usamos map41 para agrupar en una misma clave todos los pares $< testigo, cantCasosTestigoInvolu$
Luego, reduce41 calcula la máxima cantidad de casos en los que algún testigo estuvo involcrado, y devuelve un objeto con esa cantidad, $valMax$ y con la lista entera de pares $< testigo, cantCasosTestigoInvolucrado >$.

En el segundo mapeo, con map42 emitimos para una misma clave, de la lista de pares, solamente los testigos para los cuales la cantidad de casos en los que están involucrados es igual a la cantidad máxima registrada ($valMax$). Finalmente, el reduce41, se ocupa de devolver todos los testigos filtrados por map42.

```
var map41 = function() {
    emit (1, {
        testigo: this . dniTestigo,
        valor: this . idCasosInvolucradoComoTestigo . length
    });
};
var reduce41 = function(key, values) {
    id_max = values[0].testigo;
    val_max = values[0].valor;
    for(var i=1; i<values.length; i++) {
        if(val_max < values[i].valor) {
            id_max = values[i].testigo;
            val_max = values[i].valor;
        }
    }
    return {
        valMax: val_max,
        valor: values
    };
};
var map42 = function() {
    for(var i=0; i<this . value . valor . length; i++) {
        if(this . value . valor[i] . valor == this . value . valMax) {
            emit(1,
                 this . value . valor[i] . testigo
            );
        }
    }
};
var reduce42 = function(key, values) {
    const result = {
        list: []
    };
    values.forEach((value, index) => result.list = result.list.concat(
       value));
    return result;
};
db . doc3 . mapReduce(map41, reduce41, {out:"temp4"} );
db . temp4 . mapReduce(map42, reduce42, {out:"ej4"} );
```

2.4. Documeneto 4 y consultas asociadas

Finalmente, pensamos otro diseño más, para resolver las consultas restantes. Llamamo a este **doc4**. A continuación exponemos una instancia de ejemplo.

```
    [
                {
                        idCaso: 4,
                        {
                                idEvidencias:
```

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

Guardamos en cada documento un **idCaso**, y para cada uno de estos tendremos un arreglo con las distintas evidencias en **idEvidencias** y otro con los involucrados en **dniInvolucrados**.

El JSONSchema correspondiente sería:

```
{
        "type" : object,
        "properties": {
                "idCaso" : {"type" : integer},
                "idEvidencias" : {
                        "type" : array,
                        "evidencias": {
                                "type" : object,
                                "properties": {
                                        "idEvidencia" : {"type" : integer}
                                }
                        }
                },
                "dniInvolucrados" : {
                        "type" : array,
                        "involucrados": {
                                "type" : object,
                                "properties": {
                                        "dniInvolucrado" : {"type" :
                                           integer}
                                }
                        }
                }
        }
}
```

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 qué caso se trate.

2.4.2. Mantenimiento

A diferencia del anterior, en este caso podemos pensar que es realmente muy difícil que este documento ocupe demasiado espacio en disco, 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 espacio reservado para que crezca el documento en disco, podemos utilizar una técnica similar a la ya expuesta en la querie anterior.

2.4.3. Map-Reduce para casos con mayor número de involucrados

Para resolver esta consulta, empleamos dos map-reduce.

En el primero, la función map51 mapea sobre una misma clave, todos los pares $< idCaso, cantidadInvolucrados >$ que se pueden obtener del documento. Luego, reduce51 se ocupa de detectar cuál es la máxima cantidad de involucrados, y devuelve un objeto con esa máxima cantidad $cantMax$ junto con el vector con todos los pares extraídos del documento.

Luego, en map52 se agrupa en una misma clave todos los casos cuya cantidad de involucrados es igual a la máxima registrada. Por último, el reduce52 se encarga de simplemente devolver un arreglo con todos los valores resultantes del filtro aplicado por map52.

```
var map51 = function() {
    emit(1, {
        idCaso: this . idCaso,
        cantidad: this . dniInvolucrados . length
    });
};
var reduce51 = function(key, values) {
    id_max = values[0].idCaso;
    val_max = values[0].cantidad;
    for(var i=1; i<values.length; i++) {
        if(val_max < values[i].cantidad) {
            id_max = values[i].idCaso;
            val_max = values[i].cantidad;
        }
    }
    return {
        cantMax: val_max,
        valor: values
    };
};
var map52 = function() {
    for(var i=0; i<this . value . valor . length; i++) {
        if(this . value . valor[i] . cantidad == this . value . cantMax) {
            emit(1, this . value . valor[i] . idCaso);
        }
    }
};
var reduce52 = function(key, values) {
    const result = {
        list: []
    };
    values.forEach((value, index) => result.list = result.list.concat(
       value));
    return result;
};
db . doc4 . mapReduce(map51, reduce51, {out:"temp5"} );
db . temp5 . mapReduce(map52, reduce52, {out:"ej5"} );
```

2.4.4. Map-Reduce para cantidad total de evidencia por caso

Para resolver esta consulta, empleamos un único map-reduce. Map61 emite para cada caso, la cantidad de evidencias relacionadas con este. El reduce, simplemente devuelve para cada clave, el valor emitido por map61.

```
var map61 = function() {
    emit( this . idCaso, this . idEvidencias . length );
```

```
};
var reduce61 = function(key, values) {
    const result = {
        list: []
    };
    values.forEach((value, index) => result.list = result.list.concat(
       value));
    return result;
};
db . doc4 . mapReduce(map61, reduce61, {out:"ej6"} );
```

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.