

MÁSTER: “Ciencia de Datos e Ingeniería de Computadores”



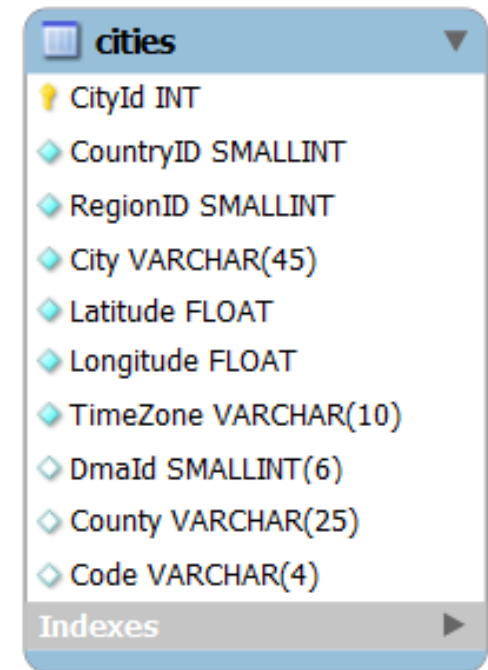
Universidad de Granada



Asignatura: Big Data y Cloud Computing

Tutorial MapReduce. Planteamiento

- Vamos a utilizar la base de datos libre [GeoWorldMap](#) de GeoBytes. Es una base de datos de [países](#), con sus [estados/regiones](#) y [ciudades](#) importantes. El esquema de la tabla ciudades es:
- Sobre esta Base de datos vamos a obtener [el par de ciudades](#) que se encuentran [más cercanas](#) en [cada país](#), [excluyendo](#) a los EEUU.
- Para ello vamos a cotejar [dos enfoques](#): [SQL](#) sobre una representación relacional de esa BD y [MapReduce](#) sobre [MongoDB](#).
- Para cada [ciudad](#) la BD almacena sus coordenadas geográficas en términos de [latitud](#) y [de longitud](#).
- En aras de la simplicidad, vamos a representar la tierra como un plano 2D. La [distancia](#) entre dos puntos P1 (x1, y1) y P2 (x2, y2) en un plano 2D se calcula como la [raíz cuadrada de \$\{\(x1-x2\)^2 + \(y1-y2\)^2\}\$](#) . Siendo y_i la [latitud](#) y x_i la [longitud](#).



cities	
CityId	INT
CountryID	SMALLINT
RegionID	SMALLINT
City	VARCHAR(45)
Latitude	FLOAT
Longitude	FLOAT
TimeZone	VARCHAR(10)
DmaId	SMALLINT(6)
County	VARCHAR(25)
Code	VARCHAR(4)
Indexes	

Tutorial MapReduce. Aproximación SQL

1. Crear una **vista** que almacene el **cuadrado** de la **distancia** para cada **par** de **ciudades** de **cada país**, salvo **EEUU**:

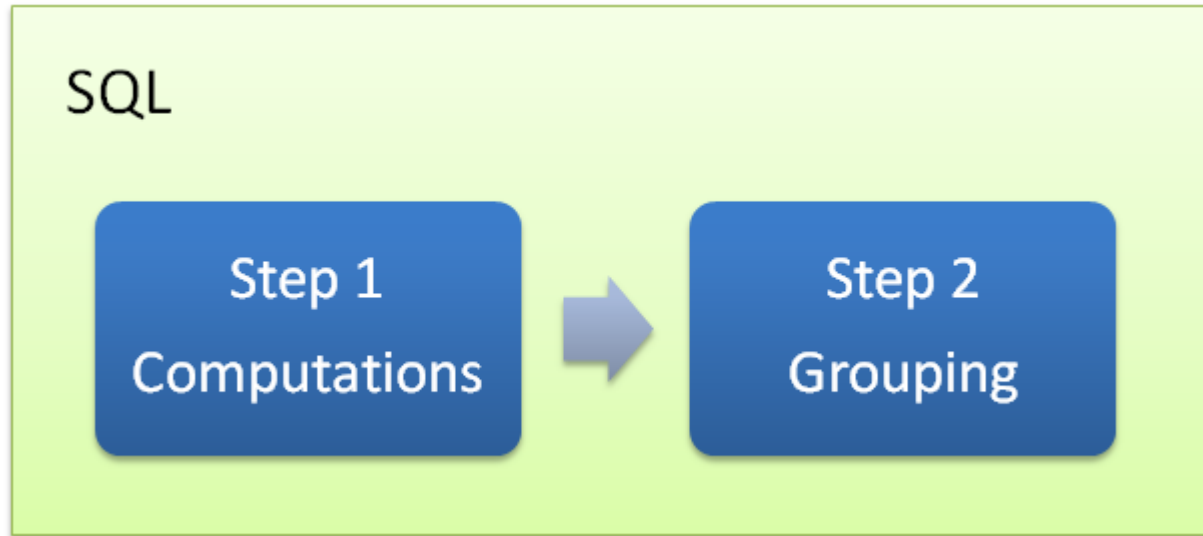
```
/* QUERY1 - VIEW: city_dist */
CREATE VIEW city_dist AS
SELECT c1.CountryID,
       c1.CityId, c1.City,
       c2.CityId AS CityId2, c2.City AS City2,
       POWER(c1.Latitude-c2.Latitude,2) +
       POWER(c1.Longitude-c2.Longitude,2) as Dist
FROM cities c1 , cities c2
WHERE c1.CountryID = c2.CountryID /* Del mismo pais*/
AND c1.CityId < c2.CityId /* Cada par de ciudades una sola vez */
AND c1.CountryID <> 254 /* No se incluyen las ciudades de EEUU */;
```

Tutorial MapReduce. Aproximación SQL

2. Agrupamos estos datos por país y seleccionamos las 2 ciudades que tienen el menor valor para el campo "Dist", siendo mayor que cero dicha distancia.

```
/* QUERY 2 */
SELECT c.CountryID,c.City,c.City2,round(sqrt(c.DIST),5) AS Distancia
FROM (
    SELECT CountryID, min(Dist) AS MinDist
    FROM city_dist
    WHERE Dist > 0 /* Evitamos ciudades que compartan latitud y
                    longitud */
    GROUP BY CountryID
) a ,city_dist c
WHERE a.CountryID = c.CountryID
      AND a.MinDist = c.Dist;
```

Tutorial MapReduce. Aproximación SQL



- Es importante tener en cuenta los pasos que seguimos:
 - 1) En el primer paso se realizaron todos los cálculos (calculando la **distancia** entre **cada 2 ciudades** de **cada país**).
 - 2) En el siguiente paso se **agruparon** los resultados **por país** y se **seleccionaron** aquellas **2 ciudades** en las que el valor de la **distancia** era **menor**.

Tutorial MapReduce. Aproximación MongoDB

- Vamos a **importar** en nuestra BD de MongoDB un archivo con 37245 ciudades del mundo que está en formato **csv** (`\var\tmp\Cities.csv`)

```
mongoimport -u <user> -p <clave> --db <bd>  
--collection cities --type csv --headerline  
--file /var/tmp/Cities.csv
```



- Vamos a implementar el **código** para **resolver** el problema sobre la recién creada colección mediante un enfoque **MapReduce** conforme a los pasos que se ilustran arriba.



Tutorial MapReduce. Aproximación MongoDB

1. **Map**. Se utiliza para dividir los datos en grupos en base a un valor deseado (llamado Key). Esto es similar a la Etapa 2 de la solución de SQL anterior. El paso de Map se implementa escribiendo una función en JavaScript, cuyo formato es el siguiente:

```
function /*void*/ MapCode() { }
```

- La función **Map** se invoca por **cada documento** de la **colección** como un método. Con **"this"** se puede acceder a cualquier dato del documento actual
- Otro elemento que está disponible es la función **"emit"** que dispone de dos argumentos: el **primero**, la **clave** sobre la que desea **agrupar** los datos; el **segundo** argumento son **los datos** que desea agrupar.

Tutorial MapReduce. Aproximación MongoDB

- Aspectos a considerar al escribir la función Map:
 - A. ¿Cómo queremos dividir o agrupar los datos? En otras palabras, ¿cuál es nuestra clave? Que es lo que se debe pasar como primer parámetro a la función "emit".
 - B. ¿Qué datos necesitaremos para el procesamiento subsiguiente? Esto ayuda a determinar que se incluye en el segundo parámetro de la función "emit".
 - C. En qué formato o estructura necesitaremos nuestros datos. Esto nos ayuda a refinar el segundo parámetro de la función de "emit".
 - D. En nuestro ejemplo los datos deben agruparse según el código de país: "CountryId". Así que éste será el primer parámetro de la función "emit"

```
MapCode function () { emit (this.CountryID, ...); }
```



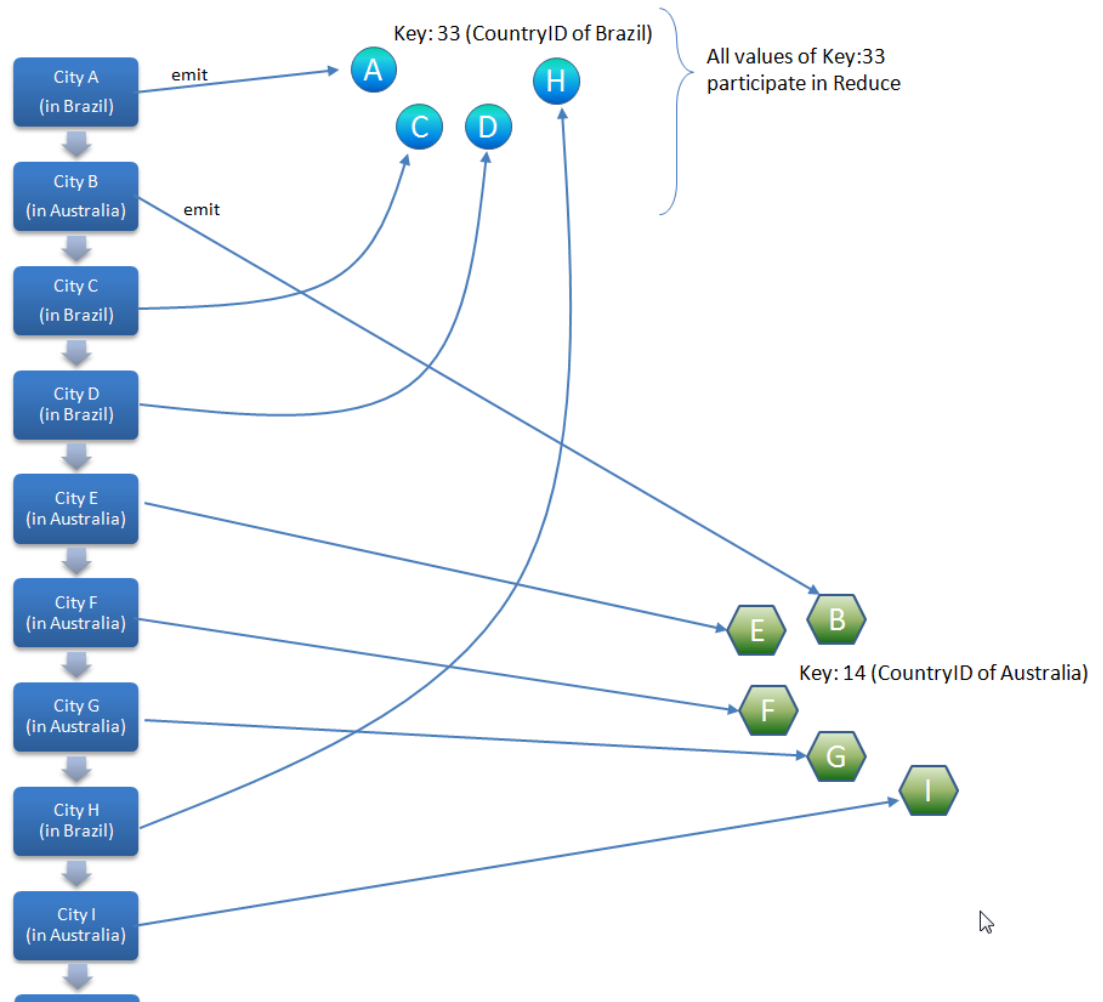
Tutorial MapReduce. Aproximación MongoDB

- B. Los **datos** que **precisaremos** para el procesamiento **posterior** serán: **City**, **Latitude** y **Longitude**. Por ello el **segundo parámetro** será un **array** que contenga todos los valores para cada ciudad de ese país:

```
function MapCode() {  
    emit(this.CountryID,  
        { "data":  
            [  
                {  
                    "city": this.City,  
                    "lat":  this.Latitude,  
                    "lon":  this.Longitude  
                }  
            ]  
        }  
    );  
}
```

Tutorial MapReduce. Aproximación MongoDB

- Después de completada la etapa **Map**, se obtienen un conjunto de pares clave-valor. En nuestro caso, en el par clave-valor la **clave** es **CountryId** y el **valor** es un **objeto JSON** como se muestra en la siguiente imagen:



Tutorial MapReduce. Aproximación MongoDB

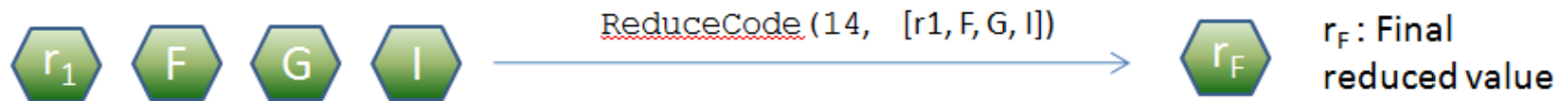
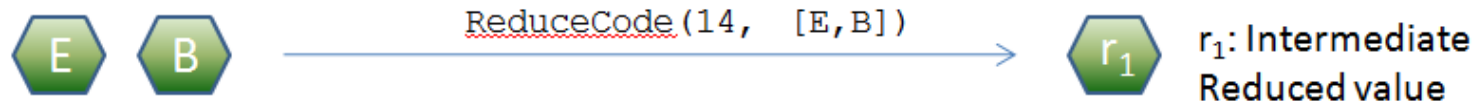
- La operación **Reduce** agrega los diferentes valores para cada clave dada usando una función definida por el usuario. En otras palabras, Reduce recorrerá cada valor de la clave (CountryId) y recogiendo todos sus valores (en nuestro caso) objetos JSON creados a partir de la etapa Map y luego los procesará uno por uno usando una lógica personalizada definida.

```
function /*object*/ ReduceCode(key, arr_values) { }
```

- Reduce** toma 2 parámetros - 1) Clave 2) un array de valores (emitidos desde el paso **Map**). La salida de **Reduce** es un objeto. Es importante tener en cuenta que **Reduce** se puede invocar varias veces desde un mismo valor de la clave. Considerar un caso en el que la cantidad de datos es enorme y se encuentran en 2 servidores diferentes. Sería ideal realizar un **Reduce** para una clave dada en el primer servidor, y luego realizar un **Reduce** para la misma clave en el segundo servidor. Y después realizar un **Reduce** sobre los resultados de estos dos valores reducidos.

Tutorial MapReduce. Aproximación MongoDB

Lets consider the Reduce of Key: 14 (Australia)



r₁ participates
in subsequent
Reduce

- No sabemos el **orden** y la **forma** en que se **aplicarían** esos **pasos** de **Reduce**, dependerá de como esté **configurado MongoDB**.
- Lo que sí sabemos es que si **Reduce** se ejecuta **más de una vez**, entonces el **valor devuelto** por la cada invocación de Reduce será **usado** en una **subsecuente invocación** de reduce como parte de la entrada de dicha función.

Tutorial MapReduce. Aproximación MongoDB

- Nuestra función **Reduce** se dedica a **integrar** todos los valores de las ciudades:

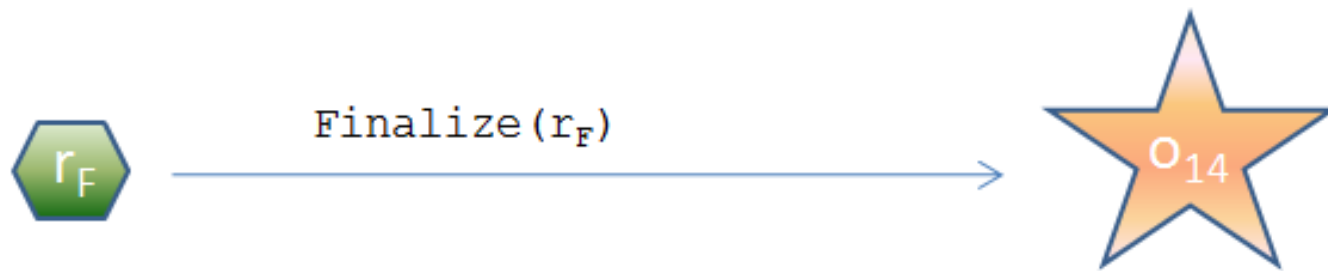
```
function ReduceCode(key, values) {  
    var reduced = {"data":[]};  
    for (var i in values) {  
        var inter = values[i];  
        for (var j in inter.data) {  
            reduced.data.push(inter.data[j]);  
        }  
    }  
    return reduced;  
}
```

Tutorial MapReduce. Aproximación MongoDB

- El siguiente **paso** es **Finalize**. **Finalize** se usa para realizar aquellas **transformaciones** que se precisen **sobre** la **salida final** de **Reduce**. La **signatura** de la función es:

```
function /*object*/ FinalizeCode(key, value) { }
```

- La función **Finalize** toma **cada par clave-valor**, y emite un objeto. La **salida** de **Finalize** para todas las claves se inserta en **una colección**, y esta colección es el **resultado del proceso MapReduce**. Se puede dar el **nombre que se desee**, y si se deja sin especificar, **MongoDB** asigna un nombre de colección.



Finalize can be used to transform the output of each Key's reduced value. The result is added to the output collection.

Tutorial MapReduce. Aproximación MongoDB

- En nuestro ejemplo usaremos **Finalize** para encontrar la **dos ciudades más próximas** en cada país.

```
function Finalize(key, reduced) {
    if (reduced.data.length == 1) {
        return { "message" : "Este pais solo contiene una ciudad" };
    }
    var min_dist = 999999999999;
    var city1 = { "name": "" };
    var city2 = { "name": "" };
    var c1;
    var c2;
    var d2;
    for (var i in reduced.data) {
        for (var j in reduced.data) {
            if (i>=j) continue;
            c1 = reduced.data[i];
            c2 = reduced.data[j];
            d2 = (c1.lat-c2.lat)*(c1.lat-c2.lat)+(c1.lon-c2.lon)*(c1.lon-c2.lon);
            if (d2 < min_dist && d2 > 0) {
                min_dist = d2;
                city1 = c1;
                city2 = c2;
            }
        }
    }
    return {"city1": city1.name, "city2": city2.name, "dist": Math.sqrt(min_dist)};
}
```

Tutorial MapReduce. Aproximación MongoDB

- Vamos **ejecutar** el comando **mapReduce** usando [db.runCommand](#)(comando) .
Cuya sintaxis general para **mapReduce** es:

```
db.runCommand(  
    {  
        mapReduce: <collection>,  
        map: <function>,  
        reduce: <function>,  
        finalize: <function>,  
        out: <output>,  
        query: <document>,  
        sort: <document>,  
        limit: <number>,  
        scope: <document>,  
        jsMode: <boolean>,  
        verbose: <boolean>  
    }  
)
```



Tutorial MapReduce. Aproximación MongoDB

```
db.runCommand({ mapReduce: "cities",
  map : function Map() {
    var key = this.CountryID;
    emit(key, {
      "data":
      [
        {
          "name" : this.City,
          "lat"  : this.Latitude,
          "lon"  : this.Longitude
        }
      ]
    });
  },
  reduce : function Reduce(key, values) {
    var reduced = {"data":[]};
    for (var i in values) {
      var inter = values[i];
      for (var j in inter.data) {
        reduced.data.push(inter.data[j]);
      }
    }
    return reduced;
  },
  ...
```



Tutorial MapReduce. Aproximación MongoDB

```
...
finalize : function Finalize(key, reduced) {
    if (reduced.data.length == 1) {
        return { "message" : "Este país sólo tiene una ciudad" };
    }
    var min_dist = 999999999999;
    var city1 = { "name": "" };
    var city2 = { "name": "" };
    var c1;
    var c2;
    var d;
    for (var i in reduced.data) {
        for (var j in reduced.data) {
            if (i>=j) continue;
            c1 = reduced.data[i];
            c2 = reduced.data[j];
            d = (c1.lat-c2.lat) * (c1.lat-c2.lat) +
                (c1.lon-c2.lon)*(c1.lon-c2.lon);
            if (d < min_dist && d > 0) {
                min_dist = d;
                city1 = c1;
                city2 = c2;
            }
        }
    }
    return { "city1": city1.name, "city2": city2.name, "dist":
Math.sqrt(min_dist)};
},
query : { "CountryID" : { "$ne" : 254 } },
out: { merge: "ciudades_proximas" }
});
```



Tutorial MapReduce. Cuestiones

1. ¿Cómo podríamos obtener la ciudades más distantes en cada país?
2. ¿Qué ocurre si en un país hay dos parejas de ciudades que están a la misma distancia mínima? ¿Cómo harías para que aparecieran todas?
3. ¿Cómo podríamos obtener adicionalmente la cantidad de parejas de ciudades evaluadas para cada país consultado?.
4. ¿Cómo podríamos la distancia media entre las ciudades de cada país?.
5. ¿Mejoraría el rendimiento si creamos un índice?¿Sobre que campo? Comprobadlo.