

## MÁSTER: “Ciencia de Datos e Ingeniería de Computadores”



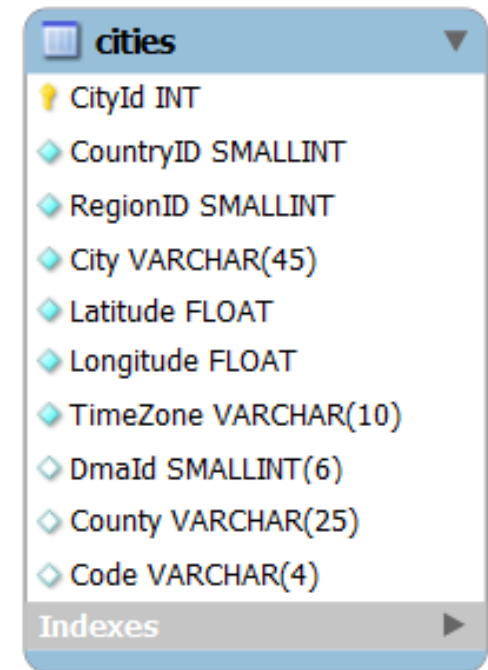
Universidad de Granada



## Big data I: Cloud Computing y Almacenamiento Masivo de Datos

# 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 [tres enfoques](#): [SQL](#) sobre una representación relacional de esa BD y [MapReduce y Aggregate](#) 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. Creamos la **tabla "cities"** mediante la siguiente sentencia:

```
CREATE TABLE CITIES
(
    "CITYID" NUMBER(*,0),
    "COUNTRYID" NUMBER(*,0),
    "REGIONID" NUMBER(*,0),
    "CITY" VARCHAR2(45 BYTE),
    "LATITUDE" FLOAT(126),
    "LONGITUDE" FLOAT(126),
    "TIMEZONE" VARCHAR2(10 BYTE),
    "DMAID" NUMBER(*,0),
    "CODE" VARCHAR2(4 BYTE)
)
```

2. **Importamos** en esa tabla la información de las ciudades **desde** el **archivo**:  
var/tmp/Cities.csv.
3. Obtenemos **la tabla con 37243** ciudades:

CITYID	COUNTRYID	REGIONID	CITY	LATITUDE	LONGITUDE	TIMEZONE	DMAID	CODE
6012	14	2	George Town	-41.1	146.833	+10:00	0	GTOW
18727	14	2	Glenorchy	-42.833	147.283	+10:00	0	GLEN
1226	14	2	Hobart	-42.917	147.333	+10:00	0	HOBA
6017	14	2	Launceston	-41.45	147.167	+10:00	0	LAUN
15999	14	2	Penguin	-41.117	146.067	+10:00	0	PENG
6025	14	2	Strahan	-42.15	145.333	+10:00	0	STRA
15650	14	2	Wynyard	-41	145.717	+10:00	0	WYNY

# Tutorial MapReduce. Aproximación SQL

```
SELECT c21.CountryID,c21.City, c22.City, round(a.Dist,5) AS Distancia
FROM (SELECT c1.CountryID as CountID ,min (sqrt(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 */
      AND (POWER(c1.Latitude-c2.Latitude,2)+POWER(c1.Longitude-c2.Longitude,2))>0
      /*Elimina las que están en el mismo sitio*/
GROUP BY c1.CountryID) a,
cities c21, cities c22
WHERE c21.CountryID = c22.CountryID /* Del mismo pais*/
      AND c21.CityId < c22.CityId /* Cada par de ciudades una sola vez */
      AND c21.CountryID <> 254 /* No se incluyen las ciudades de EEUU */
      AND c21.CountryID=a.CountID
      AND sqrt(POWER(c21.Latitude-c22.Latitude,2)+
      POWER(c21.Longitude-c22.Longitude,2))=a.Dist
ORDER BY c21.CountryID
```

1. La subconsulta **en azul** calcula y devuelve para cada país (**a.CountID**) la distancia (**a.Dist**) a la que se encuentran las ciudades más próximas. Excluye EEUU (254) y repetir parejas (**c1.CityId < c2.CityId**).
2. En la **parte en verde** se selecciona para cada país el **par de ciudades** cuya **distancia coincide con la calculada en la subconsulta** anterior y **se muestran junto con el código de país y distancia**.



# Tutorial MapReduce. Aproximación SQL



- Es importante tener en cuenta los pasos que seguimos:
  - 1) En el **primer paso** se agruparon los pares de ciudades por país y se seleccionaron las **distancias mínimas para cada país**.
  - 2) En el **siguiente paso** se **obtuvieron, para cada país, los pares de ciudades cuya distancia coincidía con los resultados anteriores y se mostraron todos los datos solicitados**.

# Tutorial MapReduce. Aproximación MongoDB

- Vamos a **importar** en nuestra BD de MongoDB un archivo con 37243 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
```

- Ahora tenemos la **colección** “cities” con 37243 documentos con el siguiente **formato**:

```
{
  "_id" : ObjectId("5704da4d7587ecbe68fd52fb"),
  "CityId" : 42231,
  "CountryID" : 1,
  "RegionID" : 833,
  "City" : "Herat",
  "Latitude" : 34.333,
  "Longitude" : 62.2,
  "TimeZone" : "+04:30",
  "DmaId" : 0,
  "Code" : "HERA"
}
```

# Tutorial MapReduce. Aproximación MongoDB



- 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). 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 desean 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".
- A. 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

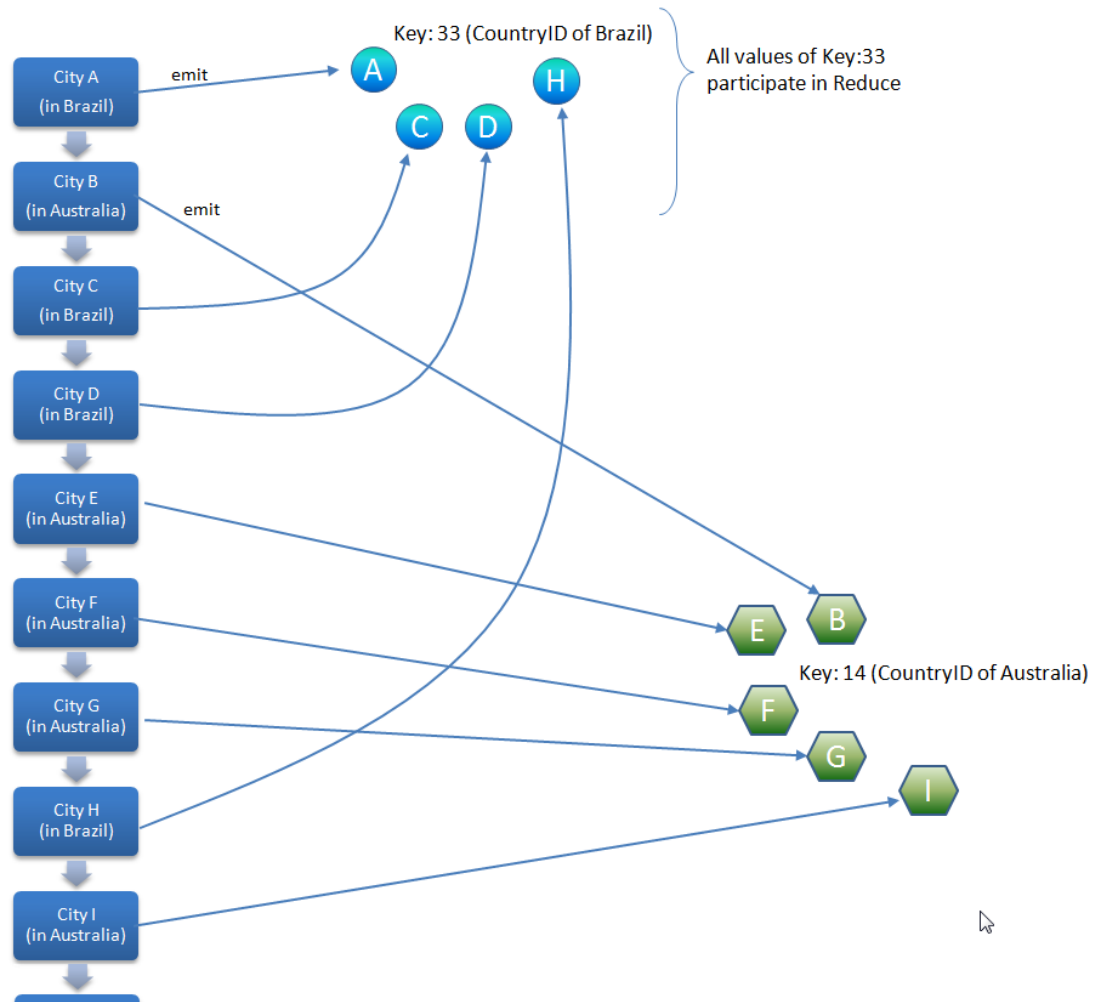
- B. Los **datos** que **precisaremos** para el procesamiento **posterior** serán: **City**, **Latitude** y **Longitude**. Por ello el **segundo parámetro** será un "array" que contenga todos los valores para cada ciudad de ese país:

```
function MapCode() {
    emit(this.CountryID,
        { "data"://contiene un array con todos los (city,lat,lon) del CountryID
          [
              {
                  "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.

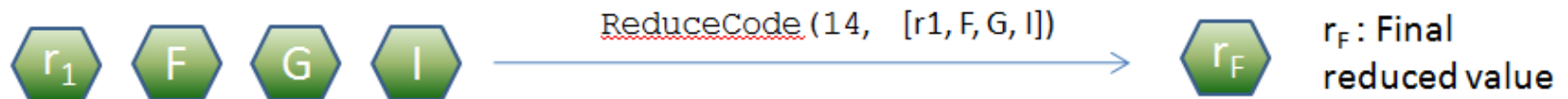
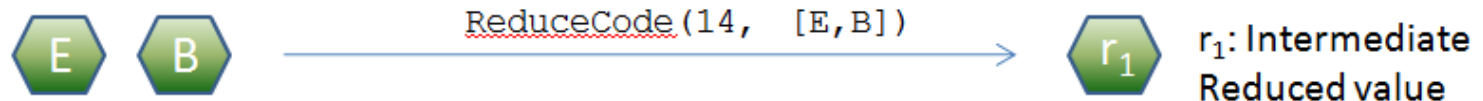
```
function /*object*/ ReduceCode(key, arr_values) { }
```

- Reduce** toma 2 parámetros: 1) Clave y 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)



$r_1$  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**, procedentes de **posibles llamadas anteriores** a la función **Reduce**, nótese que el **documento devuelto por** la función **Reduce** tiene el **mismo formato y nombre** (data: []) **que la entrada “values” de la misma**):

```
function ReduceCode(key, values) {  
    var reduced = {"data":[]};  
    for (var i in values) { /* Para cada valor en values, que puede incluir  
                            lo devuelto por una llamada anterior a reduce*/  
        var inter = values[i];  
        for (var j in inter.data) {  
            //Añade al array de la variable devuelta cada elemento  
            //del array de entrada (values)  
            reduced.data.push(inter.data[j]);  
        }  
    }  
    return reduced;  
}
```

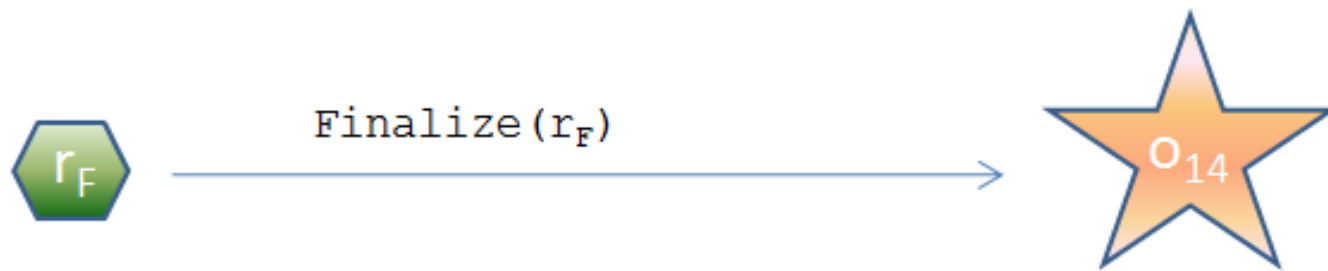


# Tutorial MapReduce. Aproximación MongoDB

- 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; //termina la iteración actual y continua con la siguiente j
            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;
  },
  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.

# Tutorial MapReduce. Mediante agregación

- Veamos como resolvemos el problema anterior usando un **enfoque basado** en el uso del operador `aggregate` y su **“pipeline”** de **agregación**:
- **Primero**, para cada país (exceptuando el 254 de EEUU), agrupamos por **CountryID**, añadiendo dos “arrays” (ciudad1 y ciudad2) conteniendo **dos veces** los datos de cada ciudad de cada país. Después, **desanidamos** esos “arrays” (`$unwind`) y, mediante `$project`, calculamos y obtenemos la **distancia** para **cada par de ciudades**, además de devolver **sus datos más relevantes**.

```
db.runCommand({
  aggregate: "cities",
  pipeline : [
    { $match: { CountryID: { $ne: 254 } } }, //Descarta ciudades de EEUU
    //Agrupar por código de país y le añade los arrays ciudad1 y ciudad2 con los datos de las ciudades de ese país
    { $group: { _id: "$CountryID", "ciudad1": { $push: { citID: "$CityId", nomcity: "$City", lat: "$Latitude", lon: "$Longitude" } },
      "ciudad2": { $push: { citID: "$CityId", nomcity: "$City", lat: "$Latitude", lon: "$Longitude" } } } },
    { $unwind: "$ciudad1" }, //Desanida ciudad1, crea un documento por cada elemento del array ciudad1
    { $unwind: "$ciudad2" }, //Desanida ciudad2, crea un documento por cada elemento del array ciudad2
    //Calcula la distancia entre cada par de ciudades en el campo "distancia", devuelve otros datos necesarios.
    { $project: { _id: 0, Pais: "$_id", ciudad1: "$ciudad1.nomcity", ciudad2: "$ciudad2.nomcity",
      distancia: { $sqrt: { $sum: [ { $pow: [ { $subtract: [ "$ciudad1.lat", "$ciudad2.lat" ] }, 2 ] },
        { $pow: [ { $subtract: [ "$ciudad1.lon", "$ciudad2.lon" ] }, 2 ] } ] } } } } },
```



# Tutorial MapReduce. Mediante agregación

- Mediante `$redact` eliminamos aquellos documentos que representan parejas de ciudades consigo mismo, parejas duplicadas, y aquellas ciudades que están a distancia 0.
- Agrupamos de nuevo por país, obteniendo la distancia mínima de cada país y añadiendo un “array” con las parejas de ciudades de ese país junto con su distancia. Desanidamos de nuevo.
- Mediante `$redact` nos quedamos con aquellos documentos que representan parejas de ciudades que cumplen que su distancia coincide con la distancia mínima del país
- Por último, proyectamos sobre los datos solicitados.

```
// Eliminamos parejas de ciudades redundantes y aquellas parejas que están a distancia 0.
{$redact: {"$cond": [{"$and":[{"$lt": ["$ciudad1", "$ciudad2"]}, {"$ne": ["$distancia", 0.0]}]}, {"$$KEEP", "$$PRUNE"}]},
{$group: {_id: "$Pais", "dist_min": {"$min": "$distancia"}, // Obtenemos la distancia mínima para cada país
          // Añadimos a la salida un "array" con los datos de todas las parejas de ciudades de cada País
          "parejas": {"$push": {"ciudad1": "$ciudad1", "ciudad2": "$ciudad2", "distancia": "$distancia"}}}},
{$unwind: "$parejas"}, // Desanidamos el "array" parejas
// Nos quedamos con aquellas parejas cuya distancia coincide con la distancia mínima de ese país
{$redact: {"$cond": [{"$eq": ["$dist_min", "$parejas.distancia"]}, {"$$KEEP", "$$PRUNE"}]},
// Proyectamos sobre los datos solicitados
{$project: {_id: 0, "CountryID": "$_id", "Ciudad1": "$parejas.ciudad1", "Ciudad2": "$parejas.ciudad2",
            "distancia": "$dist_min"}}
],
allowDiskUse: true, // Permite el uso de disco para operaciones intermedias que no quepan en memoria
cursor: { batchSize: 180 }});
```



# Tutorial MapReduce. Mediante agregación

## Resultado:

```
{
  "cursor" : {
    "id" : NumberLong(0),
    "ns" : "BigDataBase.cities",
    "firstBatch" : [
      {
        "CountryID" : 1,
        "Ciudad1" : "Kabul",
        "Ciudad2" : "Mazar-e Sharif",
        "distancia" : 3.01734618497779
      },
      {
        "CountryID" : 2,
        "Ciudad1" : "Korce",
        "Ciudad2" : "Tirane",
        "distancia" : 1.18601781183927
      },
      {
        "CountryID" : 3,
        "Ciudad1" : "Mascara",
        "Ciudad2" : "Oran",
        "distancia" : 0.836500448296355
      },
      {
        "CountryID" : 5,
        "Ciudad1" : "Andorra La Vella",
        "Ciudad2" : "Escaldes",
        "distancia" : 0.016
      }
    ], ...
  }
}
```



# Tutorial MapReduce. Consideraciones

- El tiempo de ejecución de la consulta mediante agregación es de unos 4 segundos.
- Mediante el enfoque MapReduce se ejecuta en medio segundo.
- En SQL en un segundo más o menos.
- No se ha usado paralelismo en MongoDB. Sólo tiene un “cluster”.
- A pesar de que la implementación de “aggregate” en teoría es más eficiente, por estar hecha en C++, frente al javascript usado para MapReduce, para este tipo de operación es menos eficiente.
- La necesidad de realizar dos agrupamientos y los sucesivos desanidamientos en la solución basada en “aggregate” puede que afecte al rendimiento.
- En general, las consultas que involucren un agrupamiento conjuntamente con operaciones complejas sobre los datos de cada grupo, pueden ser más eficientes usando MapReduce.
- Para el resto de las operaciones puede que “aggregate” sea más eficiente aunque hay que considerar las dos opciones, para cada caso. Véase el siguiente ejemplo:



# Tutorial MapReduce. Consulta 2

- Vamos a resolver mediante MapReduce y agregación la siguiente consulta:  
“Mostrar el identificador de País (*CountryID*) y la cantidad de *ciudades* que hay *en cada región (RegionID)*”.
- MapReduce (tiempo ejecución: 0,396 sg.)

```
db.runCommand({
  mapReduce: "cities",
  map: function mapRegion() { //Key es RegionID, value doc con CountryID y
                               //Cantidad_ciudades para calcular cuantas por region
    emit(this.RegionID, {pais: this.CountryID, Cantidad_Ciudades: 1}); },
  reduce: function reduceRegion(keyRegion, cuentas) {
    reducedVal = { PaisID: cuentas[0].pais, Cantidad_Ciudades: 0 };
    for (var idx = 0; idx < cuentas.length; idx++) {
      //Calcula cuantas ciudades hay para el valor de la clave (RegionID)
      reducedVal.Cantidad_Ciudades += cuentas[idx].Cantidad_Ciudades;
    }
    return reducedVal; },
  out: { inline: 1 }
});
```



# Tutorial MapReduce. Consulta 2

- Vamos a resolver mediante MapReduce y agregación la siguiente consulta:  
“Mostrar el identificador de País (*CountryID*) y la cantidad de *ciudades* que hay en cada región (*RegionID*)”.
- Aggregate (tiempo ejecución: 0,033 sg.) . Una *décima parte* con respecto a MapReduce:

```
db.runCommand(  
{  
  aggregate: "cities",  
  pipeline:[ //Agrupa por cod. de región y muestra cod. país y cuenta de  
             //ciudades  
    { $group: { _id: "$RegionID", PaisID: { $first: "$CountryID" },  
              Cantidad_ciudades: { $sum: 1 } } },  
    //Proyecta cod. de Pais, cod. de región y cuenta de ciudades  
    { $project: { RegionID: "$_id", "PaisID": 1, _id: 0, Cantidad_ciudades: 1 } }  
  ],  
  cursor: { batchSize: 1600 } //Cantidad de documentos mostrada en la salida  
});
```



# Tutorial MapReduce. Ejercicios

1. Resolver mediante **MapReduce** y **agregación** la siguiente consulta: “Mostrar, para **cada calle**, **cuantos** restaurantes hay”.
  2. Para la **consulta anterior**, considerar **sólo** los restaurantes *que hayan obtenido un grado “A” en cualquier momento*.
  3. Para la consulta anterior, muestra el resultado **ordenado**, de forma **decreciente**, por **cantidad de restaurantes**.
  4. Para la consulta del **ejercicio 2**, muestra la **calle** y los **nombres** de los restaurantes de la **calle que más restaurantes tenga**.
  5. Podéis comprobar que el **tiempo de ejecución mediante** el enfoque de **agregación** es unas **10 veces menor que** mediante el enfoque de **MapReduce**, además de que, en algunos ejercicios, el enfoque **MapReduce** precisa de la **creación de un relación intermedia**.
- ¿**Qué concluís** acerca de que **enfoque** es **más adecuado** para cada **tipo** de problema?.