PRACTICA 3 CLOUD COMPUTING: MONGO DB

Manuel Blanco Rienda

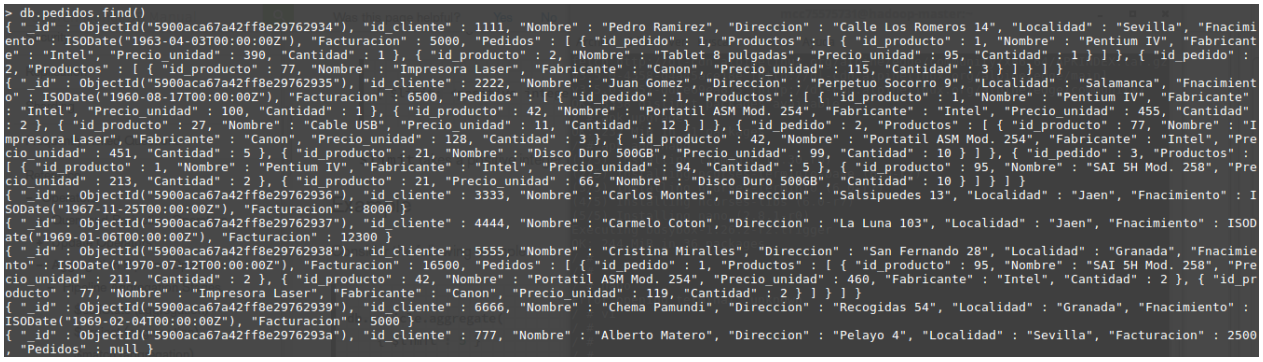
MASTER EN INGENIERÍA INFORMÁTICA manuelbr@correo.ugr.es

Objetivo Nº1: Consulta de documentos

A partir de la colección “pedidos”, se han realizado las siguientes consultas con los resultados mostrados a continuación:

1- Visualiza la colección pedidos y familiarízate con ella. Observa los distintos tipos de datos y sus estructuras dispares.

Utilizando el comando “db.pedidos.find()” he obtenido la siguiente salida, pudiendo observar la estructura de los documentos de esta colección.



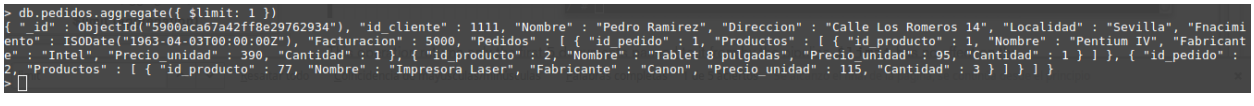
```
> db.pedidos.find()
{ "_id" : ObjectId("5900aca67a42ff8e29762934"), "id_cliente" : 1111, "Nombre" : "Pedro Ramirez", "Direccion" : "Calle Los Romeros 14", "Localidad" : "Sevilla", "Fnacimiento" : ISODate("1963-04-03T00:00:00Z"), "Facturacion" : 5000, "Pedidos" : [ { "id_pedido" : 1, "Productos" : [ { "id_producto" : 1, "Nombre" : "Pentium IV", "Fabricante" : "Intel", "Precio_unidad" : 390, "Cantidad" : 1 }, { "id_producto" : 2, "Nombre" : "Tablet 8 pulgadas", "Precio_unidad" : 95, "Cantidad" : 1 } ] }, { "id_pedido" : 2, "Productos" : [ { "id_producto" : 77, "Nombre" : "Impresora Laser", "Fabricante" : "Canon", "Precio_unidad" : 115, "Cantidad" : 3 } ] } ] }
{ "_id" : ObjectId("5900aca67a42ff8e29762935"), "id_cliente" : 2222, "Nombre" : "Juan Gomez", "Direccion" : "Perpetuo Socorro 9", "Localidad" : "Salamanca", "Fnacimiento" : ISODate("1960-08-17T00:00:00Z"), "Facturacion" : 6500, "Pedidos" : [ { "id_pedido" : 1, "Productos" : [ { "id_producto" : 1, "Nombre" : "Pentium IV", "Fabricante" : "Intel", "Precio_unidad" : 100, "Cantidad" : 1 }, { "id_producto" : 42, "Nombre" : "Portatil ASM Mod. 254", "Fabricante" : "Intel", "Precio_unidad" : 455, "Cantidad" : 2 }, { "id_producto" : 27, "Nombre" : "Cable USB", "Precio_unidad" : 11, "Cantidad" : 12 } ] }, { "id_pedido" : 2, "Productos" : [ { "id_producto" : 77, "Nombre" : "Impresora Laser", "Fabricante" : "Canon", "Precio_unidad" : 128, "Cantidad" : 3 }, { "id_producto" : 42, "Nombre" : "Portatil ASM Mod. 254", "Fabricante" : "Intel", "Precio_unidad" : 451, "Cantidad" : 5 }, { "id_producto" : 21, "Nombre" : "Disco Duro 500GB", "Precio_unidad" : 99, "Cantidad" : 10 } ] }, { "id_pedido" : 3, "Productos" : [ { "id_producto" : 1, "Nombre" : "Pentium IV", "Fabricante" : "Intel", "Precio_unidad" : 94, "Cantidad" : 5 }, { "id_producto" : 95, "Nombre" : "SAI 5H Mod. 258", "Precio_unidad" : 213, "Cantidad" : 2 }, { "id_producto" : 21, "Precio_unidad" : 66, "Nombre" : "Disco Duro 500GB", "Cantidad" : 10 } ] } ] }
{ "_id" : ObjectId("5900aca67a42ff8e29762936"), "id_cliente" : 3333, "Nombre" : "Carlos Montes", "Direccion" : "Salsipuedes 13", "Localidad" : "Jaen", "Fnacimiento" : ISODate("1967-11-25T00:00:00Z"), "Facturacion" : 8000 }
{ "_id" : ObjectId("5900aca67a42ff8e29762937"), "id_cliente" : 4444, "Nombre" : "Carmelo Coton", "Direccion" : "La Luna 103", "Localidad" : "Jaen", "Fnacimiento" : ISODate("1969-01-06T00:00:00Z"), "Facturacion" : 12300 }
{ "_id" : ObjectId("5900aca67a42ff8e29762938"), "id_cliente" : 5555, "Nombre" : "Cristina Miralles", "Direccion" : "San Fernando 28", "Localidad" : "Granada", "Fnacimiento" : ISODate("1970-07-12T00:00:00Z"), "Facturacion" : 16500, "Pedidos" : [ { "id_pedido" : 1, "Productos" : [ { "id_producto" : 95, "Nombre" : "SAI 5H Mod. 258", "Precio_unidad" : 211, "Cantidad" : 2 }, { "id_producto" : 42, "Nombre" : "Portatil ASM Mod. 254", "Precio_unidad" : 460, "Fabricante" : "Intel", "Cantidad" : 2 }, { "id_producto" : 77, "Nombre" : "Impresora Laser", "Fabricante" : "Canon", "Precio_unidad" : 119, "Cantidad" : 2 } ] } ] }
{ "_id" : ObjectId("5900aca67a42ff8e29762939"), "id_cliente" : 6666, "Nombre" : "Chema Pamundi", "Direccion" : "Recogidas 54", "Localidad" : "Granada", "Fnacimiento" : ISODate("1969-02-04T00:00:00Z"), "Facturacion" : 5000 }
{ "_id" : ObjectId("5900aca67a42ff8e2976293a"), "id_cliente" : 777, "Nombre" : "Alberto Matero", "Direccion" : "Pelayo 4", "Localidad" : "Sevilla", "Facturacion" : 2500, "Pedidos" : null }
```

Un ejemplo que muestra la arquitectura de los documentos de la colección de pedidos es el siguiente (incluido en la base de datos):

```
{ "_id" : ObjectId("5900aca67a42ff8e29762934"),
    "id_cliente" : 1111,
    "Nombre" : "Pedro Ramirez",
    "Direccion" : "Calle Los Romeros 14",
    "Localidad" : "Sevilla",
    "Fnacimiento" : ISODate("1963-04-03T00:00:00Z"),
    "Facturacion" : 5000,
    "Pedidos" : [
            {"id_pedido" : 1,
            "Productos" : [
                {"id_producto" : 1,
                "Nombre" : "Pentium IV",
                "Fabricante" : "Intel",
                "Precio_unidad" : 390,
                "Cantidad" : 1 },

                {"id_producto" : 2,
                "Nombre" : "Tablet 8 pulgadas",
                "Precio_unidad" : 95,
                "Cantidad" : 1 } ] },

            {"id_pedido" : 2,
            "Productos" : [
                { "id_producto" : 77,
                  "Nombre" : "Impresora Laser",
                  "Fabricante" : "Canon",
                  "Precio_unidad" : 115,
                  "Cantidad" : 3 } ] }
    ]
}
```

2- Visualiza sólo el primer documento de la colección. Utiliza los métodos “.limit()” ó “.findOne()”.

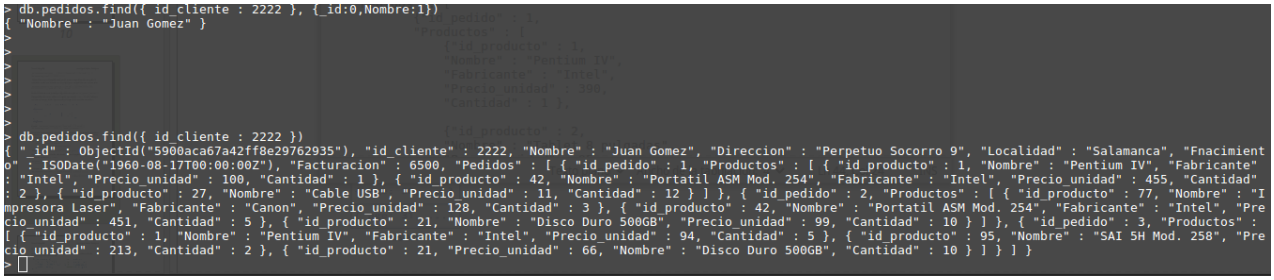
A través del uso del comando: “db.pedidos.aggregate({ $limit : 1 })”, logro obtener el primer documento de la colección, haciendo uso de la función limit con el pipeline de agregación, tal y como se usará en el objetivo 2.



```
> db.pedidos.aggregate({ $limit: 1 })
{ "_id" : ObjectId("5900aca67a42ff8e29762934"), "id_cliente" : 1111, "Nombre" : "Pedro Ramirez", "Direccion" : "Calle Los Romeros 14", "Localidad" : "Sevilla", "Fnacimiento" : ISODate("1963-04-03T00:00:00Z"), "Facturacion" : 5000, "Pedidos" : [ { "id_pedido" : 1, "Productos" : [ { "id_producto" : 1, "Nombre" : "Pentium IV", "Fabricante" : "Intel", "Precio_unidad" : 390, "Cantidad" : 1 }, { "id_producto" : 2, "Nombre" : "Tablet 8 pulgadas", "Precio_unidad" : 95, "Cantidad" : 1 } ] }, { "id_pedido" : 2, "Productos" : [ { "id_producto" : 77, "Nombre" : "Impresora Laser", "Fabricante" : "Canon", "Precio_unidad" : 115, "Cantidad" : 3 } ] } ] }
>
```

3- Visualiza el cliente con id_cliente = 2222.

Se ha usado un simple filtrado a partir de la operación find, mostrando primero sólo el nombre del cliente identificado con el número 2222 y luego mostrando todos los datos relativos al documento donde aparece (pedido que realizó y datos personales). El comando es: “db.pedidos.find({ id_cliente : 2222 }, {_id:0, Nombre : 1})” (para mostrar solo el nombre del cliente. Para mostrar todo el documento quitamos del comando el segundo bloque con llaves).



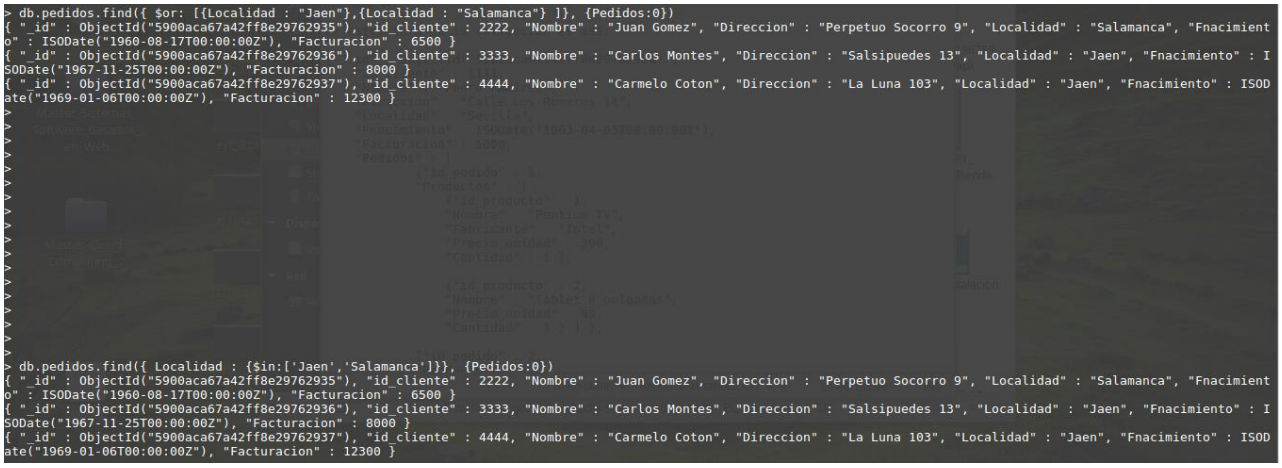
4- Visualiza los clientes que hayan pedido algún producto de más de 94 euros.

Se ha usado un filtrado a partir de la operación find combinado con el uso de operadores de comparación (“mayor que ($gt)”), mostrando primero sólo el nombre del cliente identificado y luego mostrando todos los datos relativos al documento donde aparece (pedido que realizó y datos personales). El comando es: “db.pedidos.find({ “Pedidos.Productos.Precio_unidad” : {$gt : 94} }, {_id : 0, Nombre: 1})” (para mostrar solo el nombre del cliente. Para mostrar todo el documento quitamos del comando el segundo bloque con llaves).

5- Visualiza los clientes de Jaén o Salamanca (excluye los datos de los pedidos). Utiliza los operadores $or e $in.

Al igual que en los anteriores puntos, se ha hecho uso del operador find, pero esta vez combinado con el uso del operador booleano “$or” y alternativamente, con el “$in” (se ha comprobado de las dos formas, para verificar que se obtienen los mismos resultados). Se han omitido en ambos casos los datos de Pedidos, de la misma forma que antes ocultaba todo menos los nombres de los clientes. El comando usado es:

“db.pedidos.find({ $or: [{ Localidad : “Jaen” },{ Localidad : “Salamanca” }] }, {Pedidos: 0})” cuando se ha usado el operador “or” y: “db.pedidos.find({ Localidad : {$in : [‘Jaen’,’Salamanca’]} }, {Pedidos : 0})”.



6- Visualiza los clientes que no tienen campo “pedidos”.

También en este punto se ha usado el operador find, pero combinado con la opción “exists”, para obtener aquellos clientes cuyo campo “pedidos” no existe. Se han mostrado los resultados en forma de sólo los nombres de los clientes y en forma completa de sus documentos respectivos. El comando usado ha sido: “db.pedidos.find({Pedidos : {$exists : false} }, {_id : 0, Nombre : 1})”, para mostrar sólo el nombre de los clientes identificados (Para mostrar toda la información de sus documentos, eliminar el último bloque de llaves del comando).

7- Visualiza los clientes que hayan nacido en 1963.

Para lograr esto, he usado el operador find, combinado con el uso de las operaciones de comparación “mayor o igual que” ($gte) y “menor que” ($lt), utilizando los límites que concuerdan con todo el año 1963: desde el 1 de Enero de ese año, hasta el 31 de Diciembre del mismo (haciendo uso del tipo de datos ISODate). Se ha mostrado el resultado filtrado por Nombre y sin filtrar. El comando usado es: “db.pedidos.find({ Fnacimiento: {$gte : new ISODate(“1963-01-01”), $lt : new ISODate(“1964-01-01”)} }, {_id: 0, Nombre : 1})” })”, para mostrar sólo el nombre de los clientes identificados

(Para mostrar toda la información de sus documentos, eliminar el último bloque de llaves del comando).

```
> db.pedidos.find({ Fnacimiento: {$gte : new ISODate("1963-01-01"),$lt: new ISODate("1964-01-01")} },{_id: 0,Nombre : 1})
{ "Nombre" : "Pedro Ramirez" }
>
> db.pedidos.find({ Fnacimiento: {$gte : new ISODate("1963-01-01"),$lt: new ISODate("1964-01-01")} })
{ "_id" : ObjectId("5900aca67a42ff8e29762934"), "id_cliente" : 1111, "Nombre" : "Pedro Ramirez", "Direccion" : "Calle Los Romeros 14", "Localidad" : "Sevilla", "Fnacimiento" : ISODate("1963-04-03T00:00:00Z"), "Facturacion" : 5000, "Pedidos" : [ { "id_pedido" : 1, "Productos" : [ { "id_producto" : 1, "Nombre" : "Pentium IV", "Fabricante" : "Intel", "Precio_unidad" : 390, "Cantidad" : 1 }, { "id_producto" : 2, "Nombre" : "Tablet 8 pulgadas", "Precio_unidad" : 95, "Cantidad" : 1 } ] }, { "id_pedido" : 2, "Productos" : [ { "id_producto" : 77, "Nombre" : "Impresora Laser", "Fabricante" : "Canon", "Precio_unidad" : 115, "Cantidad" : 3 } ] } ] }
>
```

8- Visualiza los clientes que hayan pedido algún producto fabricado por Canon y algún producto cuyo precio sea inferior a 15 euros.

Para lograr esto, se ha usado el operador find, combinado con el uso de la operación booleana “and” y el operador de comparación “menor que” ($lt), para expresar que quiero que los documentos obtenidos tengan un producto con el Fabricante “Canon” y otro (o él mismo) con un precio inferior a 15 euros. Se ha mostrado primero la salida filtrada por nombre del cliente con el comando: “db.pedidos.find({ $and : [{“Pedidos.Productos.Fabricante” : “Canon”}, {“Pedidos.Productos.Precio_unidad” : {$lt : 15}}] }, {_id: 0, Nombre : 1})” y luego la salida sin filtrar.

```
> db.pedidos.find({ $and:[{"Pedidos.Productos.Fabricante" : "Canon"},{"Pedidos.Productos.Precio_unidad" : {$lt : 15}} ] },{_id : 0,Nombre : 1})
{ "Nombre" : "Juan Gomez" }
>
> db.pedidos.find({ $and:[{"Pedidos.Productos.Fabricante" : "Canon"},{"Pedidos.Productos.Precio_unidad" : {$lt : 15}} ] })
{ "_id" : ObjectId("5900aca67a42ff8e29762935"), "id_cliente" : 2222, "Nombre" : "Juan Gomez", "Direccion" : "Perpetuo Socorro 9", "Localidad" : "Salamanca", "Fnacimiento" : ISODate("1960-08-17T00:00:00Z"), "Facturacion" : 6500, "Pedidos" : [ { "id_pedido" : 1, "Productos" : [ { "id_producto" : 1, "Nombre" : "Pentium IV", "Fabricante" : "Intel", "Precio_unidad" : 100, "Cantidad" : 1 }, { "id_producto" : 42, "Nombre" : "Portatil ASM Mod. 254", "Fabricante" : "Intel", "Precio_unidad" : 455, "Cantidad" : 2 }, { "id_producto" : 27, "Nombre" : "Cable USB", "Precio_unidad" : 11, "Cantidad" : 12 } ] }, { "id_pedido" : 2, "Productos" : [ { "id_producto" : 77, "Nombre" : "Impresora Laser", "Fabricante" : "Canon", "Precio_unidad" : 128, "Cantidad" : 3 }, { "id_producto" : 42, "Nombre" : "Portatil ASM Mod. 254", "Fabricante" : "Intel", "Precio_unidad" : 451, "Cantidad" : 5 }, { "id_producto" : 21, "Nombre" : "Disco Duro 500GB", "Precio_unidad" : 99, "Cantidad" : 10 } ] }, { "id_pedido" : 3, "Productos" : [ { "id_producto" : 1, "Nombre" : "Pentium IV", "Fabricante" : "Intel", "Precio_unidad" : 94, "Cantidad" : 5 }, { "id_producto" : 95, "Nombre" : "SAI 5H Mod. 258", "Precio_unidad" : 213, "Cantidad" : 2 }, { "id_producto" : 21, "Precio_unidad" : 66, "Nombre" : "Disco Duro 500GB", "Cantidad" : 10 } ] } ] }
>
```

9- Datos personales (id_cliente, Nombre, Dirección, Localidad y Fnacimiento) de los clientes cuyo nombre empieza por la cadena “c” (No distinguir entre mayúsculas y minúsculas).

En este punto también he usado la operación find, pero combinándolo esta vez con la función regex (con la sintaxis ‘^c’ aplicada al nombre de los clientes, para establecer que quiero obtener aquellos cuyo nombre comience por c) con la opción “i” para definir que sea irrelevante la distinción entre mayúsculas y minúsculas en la búsqueda. Además, de la misma forma que vengo haciendo desde el principio, se ha usado la sintaxis de filtrado de la operación find, pero ampliando la gama de informaciones que quiero obtener por documento. El comando usado es: “db.pedidos.find({ Nombre : {$regex : ‘^c’, $options : ‘i’} }, {_id : 0, Nombre : 1, id_cliente : 1, Direccion : 1, Localidad : 1, Fnacimiento : 1 })”.

```
> db.pedidos.find({Nombre : {$regex:'^c', $options : 'i'} },{_id : 0,Nombre : 1, id_cliente : 1, Direccion: 1, Localidad : 1, Fnacimiento: 1})
{ "id_cliente" : 3333, "Nombre" : "Carlos Montes", "Direccion" : "Salsipuedes 13", "Localidad" : "Jaen", "Fnacimiento" : ISODate("1967-11-25T00:00:00Z") }
{ "id_cliente" : 4444, "Nombre" : "Carmelo Coton", "Direccion" : "La Luna 103", "Localidad" : "Jaen", "Fnacimiento" : ISODate("1969-01-06T00:00:00Z") }
{ "id_cliente" : 5555, "Nombre" : "Cristina Miralles", "Direccion" : "San Fernando 28", "Localidad" : "Granada", "Fnacimiento" : ISODate("1970-07-12T00:00:00Z") }
{ "id_cliente" : 6666, "Nombre" : "Chema Pamundi", "Direccion" : "Recogidas 54", "Localidad" : "Granada", "Fnacimiento" : ISODate("1969-02-04T00:00:00Z") }
>
```

10- Visualiza los datos personales de los clientes (excluyendo _id). Limita los documentos a 4.

Al igual que se ha hecho anteriormente, se ha filtrado la salida del operador find (a secas sin combinarlo con operadores de selección) con los atributos mencionados por el enunciado, haciendo uso del operador limit para limitar el campo de actuación a cuatro documentos. El comando es: “db. Pedidos.find({}, {_id: 0, Nombre : 1, id_cliente : 1, Direccion : 1, Localidad : 1, Fnacimiento : 1}).limit(4)”.

```
> db.pedidos.find({},{_id : 0,Nombre : 1, id_cliente : 1, Direccion: 1, Localidad : 1, Fnacimiento: 1}).limit(4)
{ "id_cliente" : 1111, "Nombre" : "Pedro Ramirez", "Direccion" : "Calle Los Romeros 14", "Localidad" : "Sevilla", "Fnacimiento" : ISODate("1963-04-03T00:00:00Z") }
{ "id_cliente" : 2222, "Nombre" : "Juan Gomez", "Direccion" : "Perpetuo Socorro 9", "Localidad" : "Salamanca", "Fnacimiento" : ISODate("1960-08-17T00:00:00Z") }
{ "id_cliente" : 3333, "Nombre" : "Carlos Montes", "Direccion" : "Salsipuedes 13", "Localidad" : "Jaen", "Fnacimiento" : ISODate("1967-11-25T00:00:00Z") }
{ "id_cliente" : 4444, "Nombre" : "Carmelo Coton", "Direccion" : "La Luna 103", "Localidad" : "Jaen", "Fnacimiento" : ISODate("1969-01-06T00:00:00Z") }
>
```

11- Idem anterior pero ordenando los documentos por Localidad (ascendente) e id_cliente (descendente).

Partiendo del comando anterior, se ha añadido una concatenación con el operador sort, indicando en sus parámetros que ordene los documentos de salida por orden ascendente de Localidad y descendente de id_cliente. El comando es: “db. Pedidos.find({}, {_id: 0, Nombre : 1, id_cliente : 1, Direccion : 1, Localidad : 1, Fnacimiento : 1}).limit(4).sort({ Localidad : 1, id_cliente : -1})”.

```
> db.pedidos.find({},{_id : 0,Nombre : 1, id_cliente : 1, Direccion: 1, Localidad : 1, Fnacimiento: 1}).limit(4).sort({Localidad : 1,id_cliente : -1})
{ "id_cliente" : 6666, "Nombre" : "Chema Pamundi", "Direccion" : "Recogidas 54", "Localidad" : "Granada", "Fnacimiento" : ISODate("1969-02-04T00:00:00Z") }
{ "id_cliente" : 5555, "Nombre" : "Cristina Miralles", "Direccion" : "San Fernando 28", "Localidad" : "Granada", "Fnacimiento" : ISODate("1970-07-12T00:00:00Z") }
{ "id_cliente" : 4444, "Nombre" : "Carmelo Coton", "Direccion" : "La Luna 103", "Localidad" : "Jaen", "Fnacimiento" : ISODate("1969-01-06T00:00:00Z") }
{ "id_cliente" : 3333, "Nombre" : "Carlos Montes", "Direccion" : "Salsipuedes 13", "Localidad" : "Jaen", "Fnacimiento" : ISODate("1967-11-25T00:00:00Z") }
>
```

Objetivo Nº2: Agregación

A partir de la colección anterior de pedidos, se ha realizado una serie de consultas, más complejas que las ya realizadas, haciendo uso del pipeline de agregación.

1- Número total de clientes

A través del uso del operador aggregate combinado con la opción “$count”, he establecido el resultado “Número de Clientes”. El comando usado es: “db.pedidos.aggregate({ $count : “NUMERO DE CLIENTES” })”.

```
> db.pedidos.aggregate({$count : "NUMERO DE CLIENTES"})
{ "NUMERO DE CLIENTES" : 7 }
>
```

2- Número total de clientes de Jaén

Al comando anterior se le ha añadido un bloque entre llaves, justo antes del que medía el número de resultados de la consulta. Este segundo bloque utiliza la opción “$match” para buscar los documentos que tengan el valor “Jaen” en “Localidad”. El comando es: “db.pedidos.aggregate({$match : {Localidad : “Jaen”}},{$count : “NUMERO DE CLIENTES”})”.

```
>
> db.pedidos.aggregate({$match : {Localidad : "Jaen"}},{$count : "NUMERO DE CLIENTES"})
{ "NUMERO DE CLIENTES" : 2 }
>
```

3- Facturación total de clientes por localidad

Haciendo uso del operador “aggregate” junto a la opción “$group”, establezco que los documentos de la colección se agrupen por Localidad. Además, uso la operación “$sum” para establecer que quiero que se sumen todos los campos “Facturación” de los documentos que se van agrupando. El comando usado es: “db.pedidos.aggregate({ $group : {_id : “$Localidad”, TOTAL : {$sum : “$Facturacion”}} })”

```
> db.pedidos.aggregate({ $group : {_id : "$Localidad", TOTAL : {$sum : "$Facturacion"} } })
{ "_id" : "Granada", "TOTAL" : 21500 }
{ "_id" : "Jaen", "TOTAL" : 20300 }
{ "_id" : "Salamanca", "TOTAL" : 6500 }
{ "_id" : "Sevilla", "TOTAL" : 7500 }
>
```

4- Facturación media de clientes por localidad para las localidades distintas a “Jaen” con facturación media mayor a 5000. Ordenación por Localidad descendente. Eliminar el _id y poner el nombre en mayúsculas.

Al igual que antes, parto del operador “aggregate” con la opción “match”, para encontrar los documentos cuya localidad NO es Jaén (con el operador $ne). Además, realizo la misma operación de agrupamiento que en el punto anterior, pero la operación que establezco para el conjunto de los documentos agrupados es la media de la Facturación (operador avg). Una vez calculada la facturación media (a la que se da el nombre de “FACTURACION_MEDIA”), se vuelve a producir una búsqueda (en el mismo comando) para encontrar aquellos documentos que tengan un valor para ese atributo mayor a 5000 (con opción “$gt”), ordenándolos de forma descendiente por el atributo “_id”. Por último, añado la opción “$project” al comando, junto con la opción “$toUpper” aplicada a los valores de localidad agrupados, de cara a mostrar sus nombres en mayúscula. Además en este último operador se establece que quiero ver el resultado del agrupamiento por localidades en una nueva variable “Localidad” así como la facturación media calculada.

El comando usado para este fin es el siguiente: “db.pedidos.aggregate([{$match : {Localidad : {$ne : “Jaen”}}}, {$group : {_id: “$Localidad”, FACTURACION_MEDIA : {$avg : “$Facturacion”} }}, {$match : {FACTURACION_MEDIA : {$gt : {5000}}}, {$sort: {_id : -1}}, {$project : {“Localidad” : {$toUpper : “$_id”}, _id : 0, FACTURACION_MEDIA : 1}}])”

```
> db.pedidos.aggregate([{$match : {Localidad : {$ne : "Jaen"}}},{ $group : { _id : "$Localidad", FACTURACION_MEDIA : {$avg : "$Facturacion"} } },{$match : {FACTURACION_M
EDIA : {$gt : 5000} }}, {$sort : {_id : -1} }, {$project : {"Localidad": {$toUpper: "$_id"}, _id : 0, FACTURACION_MEDIA : 1} }])
{ "FACTURACION_MEDIA" : 6500, "Localidad" : "SALAMANCA" }
{ "FACTURACION_MEDIA" : 10750, "Localidad" : "GRANADA" }
>
```

5- Calcula la cantidad total facturada por cada cliente (uso de “unwind”)

Con la misma estructura que el comando anterior, utilizo la opción “unwind” aplicada a los Productos de los pedidos, para desgranar sus datos por separado y uso “group” para agrupar los documentos por id de los clientes y su nombre, calculando el precio total pagado por cada cliente (multiplicando el precio de los productos que adquirió por su cantidad) y usando la opción “project” para mostrar el nombre del cliente, su identificador y el total que pagó.

El comando usado es: “db.pedidos.aggregate([{$unwind : "$Pedidos"}, {$unwind : "$Pedidos.Productos"}, {$group : {_id : {id_cliente: "$id_cliente", nombre : "$Nombre"}, TOTAL_CLIENTE : {$sum : {$multiply : ["$Pedidos.Productos.Precio_unidad","$Pedidos.Productos.Cantidad"] } } } }, {$project : {_id : 0, NOMBRE_COMPLETO : "$_id.nombre", IDENTIFICADOR : "$_id.id_cliente", TOTAL_CLIENTE: 1} }, {$sort: {NOMBRE_COMPLETO : -1} }])”.

```
> db.pedidos.aggregate([{$unwind : "$Pedidos"}, {$unwind : "$Pedidos.Productos"}, {$group : { _id : {id_cliente: "$id_cliente", nombre : "$Nombre"}, TOTAL_CLIENTE : {$su
m : {$multiply : ["$Pedidos.Productos.Precio_unidad","$Pedidos.Productos.Cantidad"] } }  } }, {$project : {_id : 0, NOMBRE_COMPLETO : "$_id.nombre", IDENTIFICADOR : "$_
id.id_cliente", TOTAL_CLIENTE: 1} }, {$sort: {NOMBRE_COMPLETO : -1} }])
{ "TOTAL_CLIENTE" : 830, "NOMBRE_COMPLETO" : "Pedro Ramirez", "IDENTIFICADOR" : 1111 }
{ "TOTAL_CLIENTE" : 6327, "NOMBRE_COMPLETO" : "Juan Gomez", "IDENTIFICADOR" : 2222 }
{ "TOTAL_CLIENTE" : 1580, "NOMBRE_COMPLETO" : "Cristina Miralles", "IDENTIFICADOR" : 5555 }
>
```

Objetivo Nº3: Map Reduce

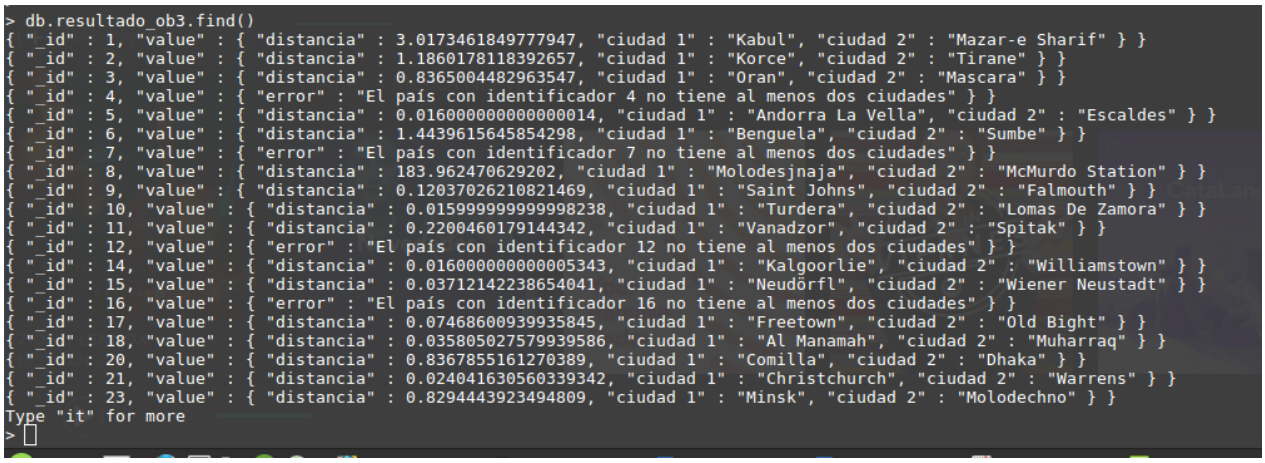
Una vez cargada la base de datos de ciudades en mongo, paso a desarrollar las funciones de mapeado, reducción y finalización que se describen en el enunciado. A continuación puede observarse el código de éstas:

```
/*****************************************************************************************************
Función de mapeado: Cada documento de la colección se divide en: clave y valor. La clave es el countryID del
país al que pertenece y el valor es un vector que tiene la información de la ciudad (nombre, longitud y latitud).
*****************************************************************************************************/

var MapCode = function () {
  emit(this.CountryID,
        {"ciudad":[
                    {
        "nombre": this.City,
        "latitud":  this.Latitude,
        "longitud":  this.Longitude
                    }]
  });
};
```

```
/*****************************************************************************************************************
Función de reducción: Se recorren todas las ciudades para cada clave de cada país. Dado que el proceso
de agrupamiento por clave ya lo hace mongo por defecto, la entrada a la función de reducción es el countryid
de cada país y los valores son los vectores con (nombre, longitud y latitud) de cada ciudad. El proceso de
agrupamiento consiste en añadir todas las ciudades que provienen del mismo país (clave) a un vector.
*****************************************************************************************************************/
var ReduceCode = function(key, arr_values){
  var mapeado = {"ciudad":[]};

        for (var i in arr_values) {
                var aux = arr_values[i];
                for (var j in aux.ciudad) {
                        mapeado.ciudad.push(aux.ciudad[j]);
                }
        }

  return mapeado;
};


/*****************************************************************************************************************
Función de finalización: Su funcionamiento se basa en la búsqueda del par de ciudades más cercanas para cada país,
menos para Estados Unidos. En cada país (representado por una iteración del proceso "Reduce"), se comprueba
que las dos ciudades a comparar no tengan la misma posición, ya que la colección tiene errores de este tipo por defecto.
Se calcula la distancia entre cada par de ciudades distintas como: Raíz cuadrada de (cuadrado de diferencia entre
longitudes más cuadrado de diferencia entre latitudes). Se va iterando para probar todas las combinaciones de ciudades
y voy acumulando la menor distancia hasta el momento, así como el nombre del par de ciudades al que corresponde.
*****************************************************************************************************************/
var FinalizeCode = function(key, value){
  var distancia;
  var ciudad1,ciudad2,min_distancia_c1,min_distancia_c2;
  var sentinel = 10000000000000000000000000000000000;

  if(value.ciudad.length < 2){
    return {
        "error" : "Este país no tiene al menos dos ciudades"
    };
  }

  for(var i in value.ciudad){
      for(var j in value.ciudad){
        if(j>i && ((value.ciudad[i].longitud != value.ciudad[j].longitud) || (value.ciudad[i].latitud != value.ciudad[j].latitud))){
          ciudad1 = value.ciudad[i];
          ciudad2 = value.ciudad[j];
          distancia = Math.sqrt(Math.pow((ciudad1.longitud - ciudad2.longitud),2) + Math.pow((ciudad1.latitud - ciudad2.latitud),2));

          if(distancia < sentinel){
            sentinel = distancia;
            min_distancia_c1 = ciudad1.nombre;
            min_distancia_c2 = ciudad2.nombre;
          }
        }
      }
  }

  return {
    "distancia" : sentinel,
    "ciudad 1" : min_distancia_c1,
    "ciudad 2" : min_distancia_c2
  };
}
```

```
/*****************************************************************************************************************
Ejecución del proceso de mapeado-reducción.
*****************************************************************************************************************/
db.runCommand({
    mapReduce: "cities",
    map: MapCode,
    reduce: ReduceCode,
    finalize: FinalizeCode,
    query: { CountryID: { $ne: 254 } },
    out: { merge: "resultados_ob3" }
});
```

La ejecución de estas funciones (alojadas en un documento Javascript) se ha llevado a cabo con el comando: “mongo < nombreDelArchivo.js”, obteniendo la siguiente salida al comprobar la colección resultante:



```
> db.resultado_ob3.find()
{ "_id" : 1, "value" : { "distancia" : 3.0173461849777947, "ciudad 1" : "Kabul", "ciudad 2" : "Mazar-e Sharif" } }
{ "_id" : 2, "value" : { "distancia" : 1.1860178118392657, "ciudad 1" : "Korce", "ciudad 2" : "Tirane" } }
{ "_id" : 3, "value" : { "distancia" : 0.8365004482963547, "ciudad 1" : "Oran", "ciudad 2" : "Mascara" } }
{ "_id" : 4, "value" : { "error" : "El país con identificador 4 no tiene al menos dos ciudades" } }
{ "_id" : 5, "value" : { "distancia" : 0.016000000000000014, "ciudad 1" : "Andorra La Vella", "ciudad 2" : "Escaldes" } }
{ "_id" : 6, "value" : { "distancia" : 1.4439615645854298, "ciudad 1" : "Benguela", "ciudad 2" : "Sumbe" } }
{ "_id" : 7, "value" : { "error" : "El país con identificador 7 no tiene al menos dos ciudades" } }
{ "_id" : 8, "value" : { "distancia" : 183.962470629202, "ciudad 1" : "Molodesjnaja", "ciudad 2" : "McMurdo Station" } }
{ "_id" : 9, "value" : { "distancia" : 0.12037026210821469, "ciudad 1" : "Saint Johns", "ciudad 2" : "Falmouth" } }
{ "_id" : 10, "value" : { "distancia" : 0.015999999999998238, "ciudad 1" : "Turdera", "ciudad 2" : "Lomas De Zamora" } }
{ "_id" : 11, "value" : { "distancia" : 0.2200460179144342, "ciudad 1" : "Vanadzor", "ciudad 2" : "Spitak" } }
{ "_id" : 12, "value" : { "error" : "El país con identificador 12 no tiene al menos dos ciudades" } }
{ "_id" : 14, "value" : { "distancia" : 0.016000000000005343, "ciudad 1" : "Kalgoorlie", "ciudad 2" : "Williamstown" } }
{ "_id" : 15, "value" : { "distancia" : 0.03712142238654041, "ciudad 1" : "Neudörfl", "ciudad 2" : "Wiener Neustadt" } }
{ "_id" : 16, "value" : { "error" : "El país con identificador 16 no tiene al menos dos ciudades" } }
{ "_id" : 17, "value" : { "distancia" : 0.07468600939935845, "ciudad 1" : "Freetown", "ciudad 2" : "Old Bight" } }
{ "_id" : 18, "value" : { "distancia" : 0.035805027579939586, "ciudad 1" : "Al Manamah", "ciudad 2" : "Muharraq" } }
{ "_id" : 20, "value" : { "distancia" : 0.8367855161270389, "ciudad 1" : "Comilla", "ciudad 2" : "Dhaka" } }
{ "_id" : 21, "value" : { "distancia" : 0.024041630560339342, "ciudad 1" : "Christchurch", "ciudad 2" : "Warrens" } }
{ "_id" : 23, "value" : { "distancia" : 0.8294443923494809, "ciudad 1" : "Minsk", "ciudad 2" : "Molodechno" } }
Type "it" for more
>
```

Tras esto, se plantean las siguientes cuestiones:

1- ¿Cómo podríamos obtener las ciudades más distantes en cada país?

Se podría obtener las ciudades más distantes de cada país simplemente alterando el algoritmo de búsqueda de la distancia mínima, para que haga lo contrario, buscar la distancia máxima entre pares de ciudades del mismo país (cambiando el operador de comparación en el bloque if y el valor base de la variable “sentinel” a 0).

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?

En mi caso, según mi código, si aparecieran dos pares de ciudades con la misma distancia mínima, se mantendrían las que primero hayan sido descubiertas. Para mantener todas las ciudades se podría volver a aplicar el paradigma “Map-Reduce” para agrupar aquellos pares con la misma clave (la distancia) y luego poder mostrarlos.

3- ¿Cómo podríamos obtener adicionalmente la cantidad de parejas de ciudades evaluadas para cada país consultado?

Se podría usar una variable (inicializada a valor cero) que acumulara el número de veces que se calcula una distancia entre dos ciudades (colocando el incremento de la variable dentro del "for" anidado) y devolviéndola en el "return" con un nombre adecuado al parámetro que representa.

4- ¿Cómo podríamos medir la distancia media entre las ciudades de cada país?

Acumulando el valor de las distancias calculadas en cada iteración y dividiendo valor total entre el valor de la variable calculada en el punto anterior (número de comparaciones hechas).

5- ¿Mejoraría el rendimiento si creamos un índice? ¿Sobre qué campo? Comprobadlo.

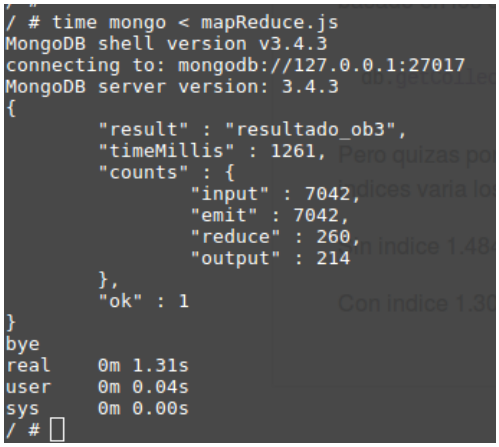
Dado que la colección con la que trabajamos es relativamente pequeña, la ejecución del procedimiento Map-Reduce no es especialmente complicada en términos de tiempo de ejecución. Es por ello que no son muy apreciables las diferencias entre usar índices y no usarlos. Los resultados de tiempo que se muestran en las diferentes configuraciones de índices son los siguientes:

Sin índices

```
/ # time mongo < mapReduce.js
MongoDB shell version v3.4.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.3
{
        "result" : "resultado_ob3",
        "timeMillis" : 1181,
        "counts" : {
                "input" : 7042,
                "emit" : 7042,
                "reduce" : 260,
                "output" : 214
        },
        "ok" : 1
}
bye
real    0m 1.23s
user    0m 0.04s
sys     0m 0.00s
```

Con índices: CountryID y CityID

```
> db.cities.ensureIndex({"CountryID" : 1, "CityId" : 1}, {unique : true})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```



También se ha probado con el atributo RegionID (tanto en conjunto con CountryID y CityId como solo), obteniendo resultados aún peores que con los mencionados índices. Es por ello que queda evidenciado que en esta colección al ser pequeña en cuanto a número de variables y parámetros, mongo es capaz de manejarse él solo de forma más eficiente con ella, que usando como índices aquellos de sus atributos más diferenciadores y representativos.