

Acceso a Datos

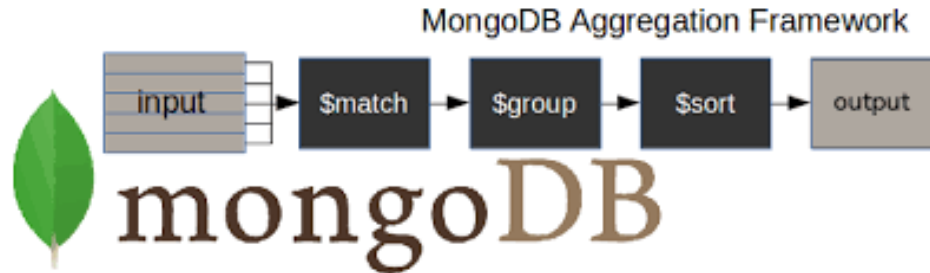
UT6 – BASES DE DATOS NOSQL

AGREGACIÓN PIPELINE

1. Agregación Pipeline



La **agregación pipeline** o tuberías de agregación se basa en someter una colección a un conjunto de **operaciones** o **etapas**. Estas etapas irán convirtiendo y transformando el conjunto de documentos pertenecientes a la colección, hasta obtener un conjunto de documentos con el resultado deseado.



1. Agregación Pipeline



Se le llama tubería ya que cada etapa irá modificando, moldeando y calculando la estructura de los documentos para pasarlos a la etapa que le sigue. Las etapas son las siguientes, y se pueden repetir hasta obtener el resultado deseado:

<u>Etapas</u>	<u>Descripción</u>	<u>Multiplicidad</u>
\$project	Cambia la forma del documento. La proyección permite modificar la representación de los datos, por lo que en general se emplea para darles una nueva forma con la que resulte más cómodo trabajar.	1:1
\$match	Filtra los resultados. La etapa match permite filtrar los documentos para que en el resultado de la etapa solo estén aquellos que cumplen ciertos criterios. Se puede filtrar antes o después de agregar los resultados, en función del orden en que definamos esta etapa	N:1
\$group	Agrupación. Permite agrupar distintos documentos según compartan el valor de uno o varios de sus atributos, y realizar operaciones de agregación sobre los elementos de cada uno de los grupos. Las funciones permitidas en la etapa group se pueden consultar aquí .	N:1

1. Agregación Pipeline



<u>Etapas</u>	<u>Descripción</u>	<u>Multiplicidad</u>
\$sort	Ordenación de documentos	1:1
\$skip	Salta N elementos	N:1
\$limit	Elige N elementos para el resultado	N:1
\$unwind	Normaliza arrays	1:N

1. Agregación Pipeline

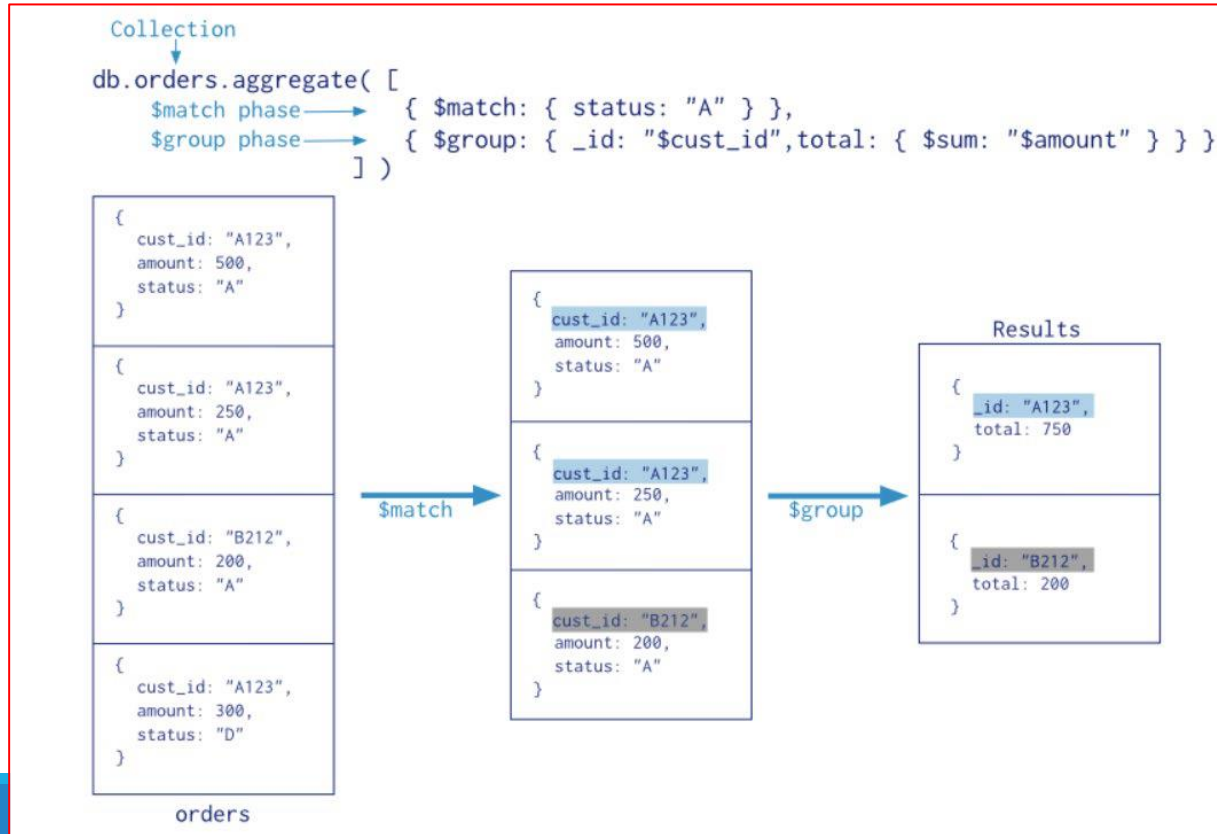


La **multiplicidad** se refiere a cuántos documentos obtenemos como resultado después de aplicar la etapa, por ejemplo, 1:1 se aplica a 1 documento y se obtiene 1. n:1 se aplica a n documentos y se obtiene 1. 1:n se aplica a un documento y se obtienen n.

Formato para utilizar las etapas:

```
db.mi_coleccion.aggregate([
  {
    $etapa1: { ... }
  },
  {
    $etapa2: {... }
  },
  .....
])
```

1. Agregación Pipeline



1. Agregación Pipeline



Ejemplos:

Para los ejemplos nos vamos a crear la colección **artículos**, que se encuentra en el archivo de colecciones MongoDB. Cada documento artículo está formado por los campos: `código`, `denominación`, `pvp` (precio de venta al público), `categoría`, `uv` (unidades vendidas), y `stock`.

1. Agregación Pipeline (a)



a) Obtener las denominaciones de los artículos y la categoría convertida a mayúsculas.

Se utiliza la etapa \$project pues cambiamos el aspecto del documento. Para referirnos a los campos del documento los ponemos entre comillas y con el prefijo \$:

```
db.articulos.aggregate(  
  [{  
    $project: {  
      denominacion: { $toUpper: "$denominacion" },  
      categoria: { $toUpper: "$categoria" }  
    }  
  }  
])
```


1. Agregación Pipeline (a)



```
< { _id: ObjectId("631a1b292e6a18213916a2b5"),  
  denominacion: 'PORTATIL ACER',  
  categoria: 'INFORMÁTICA' }  
{ _id: ObjectId("631a1b292e6a18213916a2b6"),  
  denominacion: 'PALA PáDEL',  
  categoria: 'DEPORTES' }  
{ _id: ObjectId("631a1b292e6a18213916a2b7"),  
  denominacion: 'CAJA LÁPICES',  
  categoria: 'ESCRITORIO' }  
{ _id: ObjectId("631a1b292e6a18213916a2b8"),  
  denominacion: 'MARCADORES',  
  categoria: 'ESCRITORIO' }  
{ _id: ObjectId("631a1b292e6a18213916a2b9"),  
  denominacion: 'MEMORIA 32GB',  
  categoria: 'INFORMÁTICA' }
```

1. Agregación Pipeline (a)



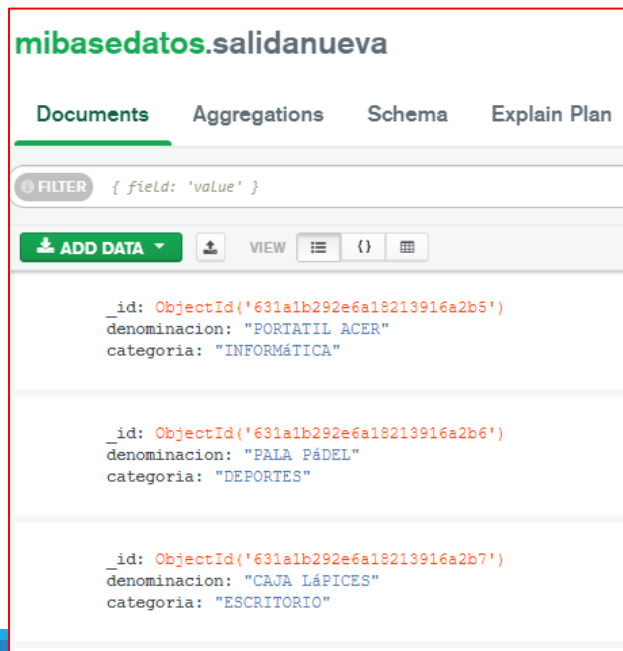
Si esta salida se desea almacenar en la base de datos, añadimos la etapa **\$out**

```
db.articulos.aggregate(  
  [{  
    $project: {  
      denominacion: { $toUpper: "$denominacion" },  
      categoria: { $toUpper: "$categoria" }  
    }  
  },  
  {  
    $out: "salidanueva"  
  }  
])
```

1. Agregación Pipeline (a)



Y como podemos comprobar se crea una nueva colección en **mibasedatos**



The screenshot shows the MongoDB Compass interface for a collection named 'mibasedatos.salidanueva'. The interface includes tabs for Documents, Aggregations, Schema, and Explain Plan. A filter bar is visible with the text '{ field: 'value' }'. Below the filter bar, there are buttons for 'ADD DATA', 'VIEW', and a dropdown menu. The main area displays three documents in a list view:

```
{
  "_id": ObjectId('631a1b292e6a18213916a2b5'),
  "denominacion": "PORTATIL ACER",
  "categoria": "INFORMÁTICA"
}
```

```
{
  "_id": ObjectId('631a1b292e6a18213916a2b6'),
  "denominacion": "PALA PáDEL",
  "categoria": "DEPORTES"
}
```

```
{
  "_id": ObjectId('631a1b292e6a18213916a2b7'),
  "denominacion": "CAJA LÁPICES",
  "categoria": "ESCRITORIO"
}
```

1. Agregación Pipeline (b)



b) Obtener la denominación en mayúsculas, el importe de las ventas ($uv * el\ pvp$), y el stock actual (stock- uv). Se utiliza la etapa `$project`, la función `$multiply` para multiplicar las uv por el pvp y `$subtract` para restar las uv del stock.

```
db.articulos.aggregate(  
  [{  
    $project:  
      { denominacion:{$toUpper:"$denominacion"},  
        importe:{$multiply:["$pvp","$uv"]},  
        stockactual:{$subtract:["$stock","$uv"]} }  
  ] ] )
```

1. Agregación Pipeline (b)



```
< { _id: ObjectId("631a1b292e6a18213916a2b5"),  
    denominacion: 'PORTATIL ACER',  
    importe: 5000,  
    stockactual: 10 }  
{ _id: ObjectId("631a1b292e6a18213916a2b6"),  
  denominacion: 'PALA PÁDEL',  
  importe: 500,  
  stockactual: 25 }  
{ _id: ObjectId("631a1b292e6a18213916a2b7"),  
  denominacion: 'CAJA LÁPICES',  
  importe: 60,  
  stockactual: -4 }  
{ _id: ObjectId("631a1b292e6a18213916a2b8"),  
  denominacion: 'MARCADORES',  
  importe: 200,  
  stockactual: -1 }
```

1. Agregación Pipeline (b)



Condiciones de agregación:

Podemos añadir las siguientes condiciones a las consultas de agregación:

- **\$cond** – Este operador evalúa una expresión y dependiendo del resultado, devuelve el valor de una de las otras dos expresiones.

- Formato:

```
{ $cond: [ <boolean-expression>, <caso-true>, <caso-false> ] }
```

- **\$ifNull** – Devuelve o bien el resultado no nulo de la primera expresión, o el resultado de la segunda expresión si la primera expresión da como resultado un resultado nulo.

- Formato:

```
{ $ifNull: [ <expression>, <expresionsiesnull> ] }
```

1. Agregación Pipeline (b)



A la consulta anterior, vamos a preguntar si el stock actual es negativo, asignaremos a un campo nuevo llamado **areponer** true si es menor que 0 y false si no lo es.

```
db.articulos.aggregate(  
  [{  
    $project:  
      { denominacion:{$toUpper:"$denominacion"},  
        importe:{$multiply:["$pvp","$uv"]},  
        stockactual:{$subtract:["$stock","$uv"]},  
        areponer:{  
          $cond:[{$lte:[$subtract:["$stock","$uv"]},0]},true,false]  
        }  
      }  
    ]  
  })
```

1. Agregación Pipeline (c)



c) En la siguiente consulta obtenemos por cada categoría el número de artículos, el total unidades vendidas de artículos, y el total importe (la suma de los $pvp * unidades$). Es como una **select** con **group by**. En este caso se utiliza la etapa **\$group**, cuando se utiliza esta etapa se debe añadir el identificador de objeto **id**, en este caso como agrupamos por categoría lo indicamos en el **_id**. Que a su vez será el identificador del resultado. Para contar artículos se utiliza la función **\$sum**, sumando 1:

1. Agregación Pipeline (c)



```
db.articulos.aggregate(  
  [{  
    $group:  
    { _id:"$categoria",  
      contador:{$sum:1},  
      sumunidades:{$sum:"$uv"},  
      totalimporte:{$sum:{$multiply:["$pvp","$uv"]}}  
    }  
  }  
]  
)
```

1. Agregación Pipeline (c)



```
< { _id: 'Escritorio',  
    contador: 2,  
    sumunidades: 30,  
    totalimporte: 260 }  
{ _id: 'Informática',  
  contador: 3,  
  sumunidades: 22,  
  totalimporte: 6560 }  
{ _id: 'Deportes',  
  contador: 3,  
  sumunidades: 30,  
  totalimporte: 725 }
```

1. Agregación Pipeline (c)



Otra forma de hacerlo sería proyectando primero para obtener el importe de cada venta

```
db.articulos.aggregate(  
  [{ $project: { importe: { $multiply: [ "$pvp", "$uv" ] },  
    categoria: "$categoria" } },  
  { $group: { _id: "$categoria",  
    totalimporte: { $sum: "$importe" } } }  
])
```

1. Agregación Pipeline (d)



d) Partiendo de la consulta anterior si queremos mostrar la categoría que mayores ventas ha tenido, tendríamos que añadir 2 etapas más: **sort** y **limit**

```
db.articulos.aggregate(  
  [{  
    $group:  
    {  
      _id:"$categoria",  
      contador:{$sum:1},  
      sumunidades:{$sum:"$uv"},  
      totalimporte:{$sum:{$multiply:["$pvp","$uv"]}}  
    }  
  },  
  { $sort:{totalimporte:-1}},  
  { $limit:1}  
]
```

1. Agregación Pipeline (e)



e) En la siguiente consulta obtenemos el número de documentos de la categoría Deportes, el total de unidades vendidas de sus artículos, el total importe y la media de unidades vendidas. Se utilizan las etapas **\$match** para seleccionar la categoría, y luego **\$group** para obtener resultados agrupados, en este caso en **_id** ponemos cualquier valor:

1. Agregación Pipeline (e)



```
db.articulos.aggregate(  
  [{  
    $match: {categoria: "Deportes"}  
  },  
  {  
    $group: {  
      _id: "deportes",  
      contador: {$sum: 1},  
      sumaunidades: {$sum: "$uv"},  
      media: {$avg: "$uv"},  
      totalimporte: {$sum: {$multiply: ["$pvp", "$uv"]}}  
    }  
  }  
])
```

```
< { _id: 'deportes',  
    contador: 3,  
    sumaunidades: 30,  
    media: 10,  
    totalimporte: 725 }
```

1. Agregación Pipeline (f)



f) En la siguiente consulta obtenemos el precio más caro, se agrupan los registros y obtenemos el máximo del pvp:

```
db.articulos.aggregate(  
  [{  
    $group: {  
      _id: null,  
      maximo: { $max: "$pvp" }  
    }  
  }  
])
```

```
< { _id: null, maximo: 500 }
```

1. Agregación Pipeline (g)



g) En la siguiente consulta obtenemos el artículo con el precio más caro. Utilizamos dos consultas:

1) Primero obtenemos los datos pvp y denominación, de todos los artículos, ordenados descendientemente (-1) por precio y denominación. Para ello utilizamos la etapa **\$sort**

2) El resultado obtenido se agrupa con **\$group**, para luego obtener el primero con la función **\$first**:

1. Agregación Pipeline (g)



```
db.articulos.aggregate(  
[  
  {$sort:{pvp:-1,denominacion:-1}},  
  {$group:  
    { _id:null,  
      mascaro:{$first:"$denominacion"},  
      precio:{$first:"$pvp"}  
    }  
  ]  
)
```

```
< { _id: null, mascaro: 'Portatil Acer', precio: 500 }
```

1. Agregación Pipeline (h)



h) En la siguiente consulta obtenemos la suma de importe de los artículos cuya denominación empieza por M o P. Para realizar esta consulta pasamos por 3 etapas:

1º. \$project: Obtenemos de todos los artículos el primer carácter de la denominación utilizando la función `$substr` y el importe de cada artículo.

2º. \$match: En la siguiente etapa se seleccionan, de los datos obtenidos en la primera etapa (`primercarac, importe`), los que tienen en `primercarac` P o M.

3º. \$group: Y finalmente se agrupa ese resultado, se añade un `id: 1`, y se suman los importes.

1. Agregación Pipeline (h)



```
db.articulos.aggregate([
  { $project:{
    primercarac:{$substr:["$denominacion",0,1]},
    importe:{$multiply:["$pvp","$uv"]}
  },
  { $match:{primercarac:{$in:['M','P','m','p']}}},
  {
    $group:{
      _id:1,
      totalimporte:{$sum:"$importe"}
    }
  }
])
```

1. Agregación Pipeline (i)



i) En la siguiente consulta obtenemos por cada categoría el artículo con el precio más caro. Para ello primero ordenamos descendientemente por categoría, pvp y denominación, utilizando la etapa **\$sort**. Y el resultado obtenido se agrupa con **\$group** para luego obtener el primero de cada categoría con la función **\$first**

```
db.articulos.aggregate([
  { $sort:{categoria:-1, pvp:-1, denominacion:-1}},
  {
    $group:{
      _id:"$categoria",
      mascaro:{$first:"$denominacion"},
      precio:{$first:"$pvp"}
    }
  }
])
```

1. Agregación Pipeline (i)



```
< { _id: 'Escritorio', mascaro: 'Marcadores', precio: 10 }  
  { _id: 'Deportes', mascaro: 'Pala Pádel', precio: 100 }  
  { _id: 'Informática', mascaro: 'Portatil Acer', precio: 500 }
```

1.1. Arrays, campos compuestos y agregados

Vamos a cargar la colección **Trabajadores** que se encuentra en el archivo de colecciones de los recursos de la unidad. El formato de un trabajador es este:

```
_id: ObjectId('631b6efb2e6a18213916a2c1')
  nombre: Object
    nomb: "Alicia"
    ape1: "Ramos"
    ape2: "Martín"
  direccion: Object
    poblacion: "Madrid"
    calle: "Avda Toledo 10"
  salario: 1200
  oficios: Array
    0: "Profesora"
    1: "Analista"
  > primas: Array
  edad: 50
```

1.1. Arrays, campos compuestos y agregados

Observa que el trabajador está formado por dos campos compuestos: nombre y dirección, y dos arrays: oficios y primas. Para acceder a los campos compuestos navegamos como si fuesen un objeto: `nombre.nomb`, o `direccion.poblacion`.

Por ejemplo: Para mostrar los trabajadores cuya población sea Toledo, la consulta sería:
`db.trabajadores.find({"direccion.poblacion":"Toledo"})`

1.1. Arrays, campos compuestos y agregados

Para trabajar con los arrays tenemos las siguientes funciones:

<u>Nombre</u>	<u>Descripción</u>
\$arrayElemAt	Devuelve el elemento especificado en el índice
\$elemMatch	Devuelve documentos que coincidan con las condiciones impuestas a objetos dentro de arrays
\$concatArrays	Devuelve un array concatenado en una cadena
\$filter	Selecciona elementos de un array y devuelve otro array con esos elementos
\$isArray	Determina si el operando es un array o no. Devuelve true o false.
\$size	Devuelve el número de elementos del array
\$slice	Devuelve un sub-set de elementos del array, se especifica el número.

1.1. Arrays, campos compuestos y agregados

Ejemplos:

a) Muestra la población, el nombre descompuesto en nombre, ape1 y ape2, el primer oficio del array oficios, el segundo oficio y el último. Si no los tiene no devuelve nada. Ordenados por población ascendentemente.

1.1. Arrays, campos compuestos y agregados

```
db.trabajadores.aggregate(  
  [ {  
    $project:{  
      poblacion:"$direccion.poblacion",  
      nombre:"$nombre.nomb",  
      apel:"$nombre.apel",  
      ape2:"$nombre.ape2",  
      oficio1:{$arrayElemAt:["$oficios",0]},  
      oficio2: {$arrayElemAt:["$oficios",1]},  
      oficioultimo: {$arrayElemAt:["$oficios",-1]},  
    },  
  },  
  {  
    $sort:{poblacion:1}  
  } ] )
```

1.1. Arrays, campos compuestos y agregados

b) Muestra el nombre y apellidos de aquellos trabajadores cuyo alguno de sus oficios empiece por C:

```
db.trabajadores.aggregate(  
  [  
    {  
      $match: {oficios: {$elemMatch: {$regex: "^C"}}}  
    },  
    {  
      $project: {nombre: "$nombre.nomb",  
                  apel1: "$nombre.apel1",  
                  ape2: "$nombre.ape2",  
                  oficios: "$oficios"  
                }  
    }  
  ]  
)
```

1.1. Arrays, campos compuestos y agregados

- **c)** Muestra el tamaño de los arrays de los trabajadores (oficios y primas), y concatenado su contenido. Se utiliza la función "\$ifNull" para comprobar que los arrays existan en los trabajadores, y evitar errores de salida ("The argument to \$size must be an array, but was of type: missing"). Se pregunta si el array es nulo, si lo es, devuelve el array vacío, el tamaño del array vacío será 0.

1.1. Arrays, campos compuestos y agregados

```
db.trabajadores.aggregate([
{
  $project:{
    nombre:"$nombre.nomb",
    numoficios:{$size:{"$ifNull":["$oficios",[]]}},
    numprimas:{$size:{"$ifNull":["$primas",[]]}},
    oficios:{$concatArrays:["$oficios","$primas"]}
  }
}] )
```

1.1. Arrays, campos compuestos y agregados

d) La siguiente consulta de agregado devuelve el número de trabajadores y la media de edad de los trabajadores que han tenido el oficio de Analista:

```
db.trabajadores.aggregate([
{ $match:{oficios:"Analista"}},
{
  $group:{
    _id:"analista",
    contador:{$sum:1},
    media:{$avg:"$edad"}
  }
}] )
```

1.1. Arrays, campos compuestos y agregados

Las fases de un pipeline, se pueden repetir y elegir el orden que más nos convenga. En el siguiente ejemplo se aplica 2 veces la etapa match:

```
db.trabajadores.aggregate([
  {$match:{"edad":{"$gt":25}}},
  {
    $group:{
      _id:"$direccion.poblacion",
      trabajadores:{$sum:1},
      media:{$avg:"$salario"}
    }
  },
  {$match:{"media":{"$gt":1300}}}
])
```

Dudas y preguntas

