



U N I V E R S I D A D  
**COMPLUTENSE**  
M A D R I D

# Tuberías de agregación *(Aggregation Pipelines)*

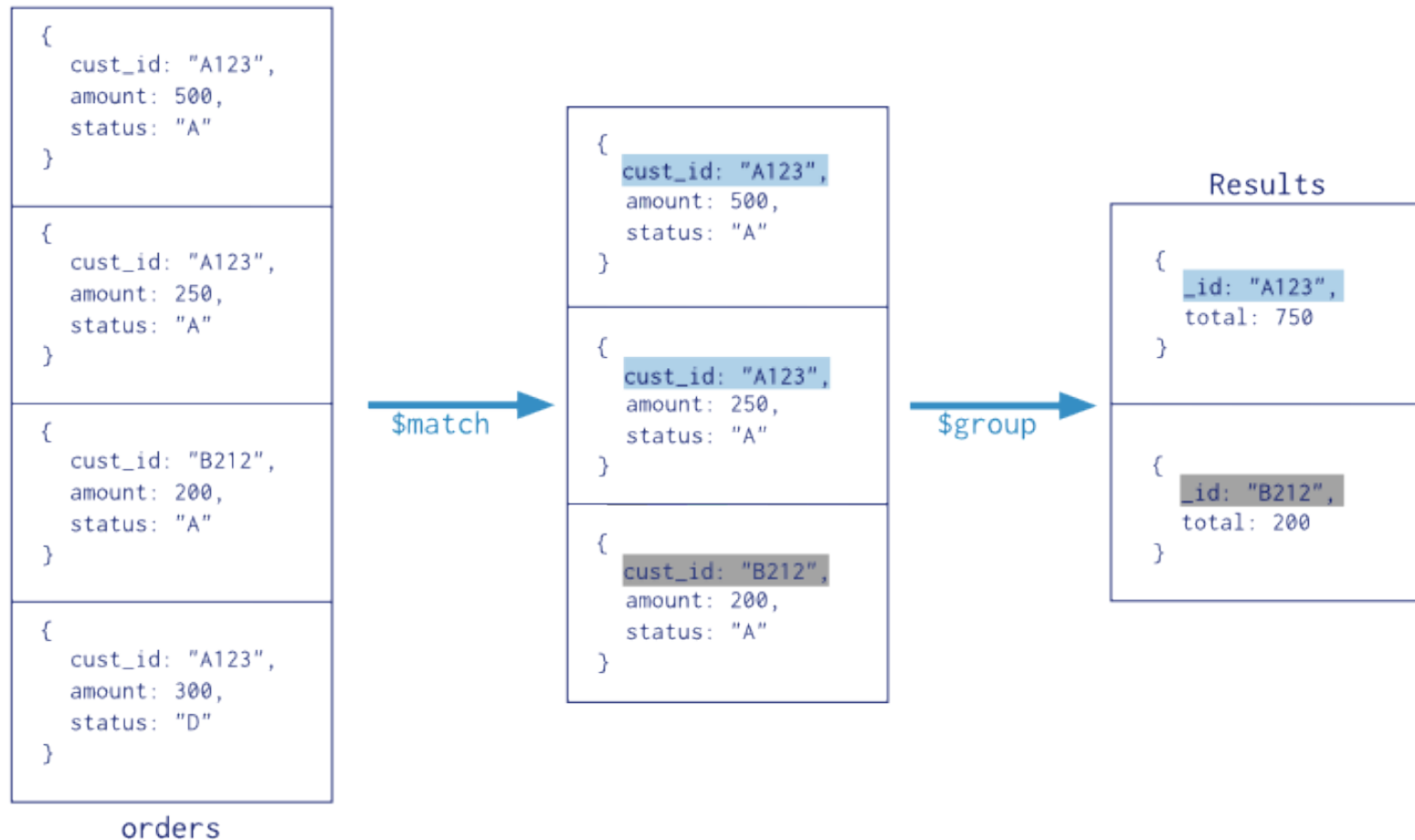
Gestión de la Información en la Web  
Enrique Martín - emartinm@ucm.es  
Grados de la Fac. Informática

# Tuberías de agregación

- Las **tuberías de agregación** (*aggregation pipelines*) son un mecanismo muy potente para procesar las colecciones aplicando varias **etapas**.
- Cada etapa recoge los datos de la etapa anterior, y su resultado es la entrada de la etapa siguiente.
- El resultado de una tubería de agregación suelen ser valores **agregados** a partir de colecciones (resúmenes, combinaciones, etc.)

# Tuberías de agregación

Collection  
↓  
db.orders.aggregate( [  
 \$match stage → { \$match: { status: "A" } },  
 \$group stage → { \$group: { \_id: "\$cust\_id", total: { \$sum: "\$amount" } } }  
] )



# Etapas

- Las tuberías de agregación pueden tener cualquier número de etapas.
- El programador configura la secuencia de etapas a aplicar mediante documentos JSON.
- El **resultado** de una tubería de agregación se puede volcar en una **nueva colección**, o devolver un cursor como si se tratase de una consulta común.
- En las **colecciones intermedias** no es necesario garantizar que el campo **\_id** sea único.

# Lanzar una tubería de agregación

- Las tuberías de agregación se lanzan con el comando **aggregate** a partir de una colección:

`db.collection.aggregate([etapa1, ..., etapan])`

- El comando **aggregate** recibe una lista de documentos **etapa<sub>1</sub>, ..., etapa<sub>n</sub>** que definen el **orden** y la **configuración** de cada una de las etapas.

# Tipos de etapas

- Existen al menos 13 tipos diferentes de etapas que se pueden utilizar (más información en las referencias).
- Veremos con detalle únicamente algunas de ellas:
  - \$project
  - \$match
  - \$sort
  - \$limit, \$skip
  - \$unwind
  - \$group, \$lookup
  - \$out

**Etapas**

# \$project

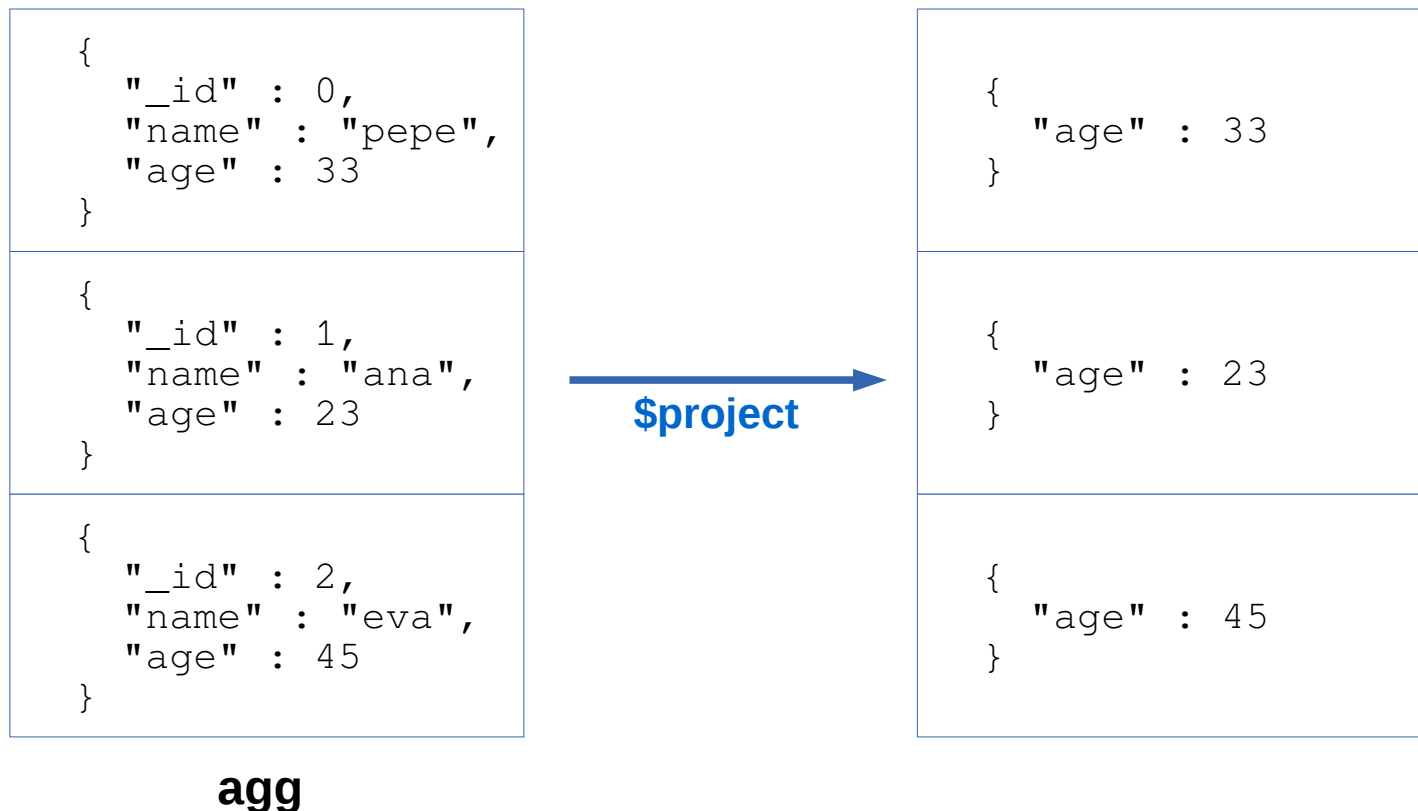
- La etapa **\$project** sirve para **eliminar o añadir campos** a los documentos, es decir, reestructurar documentos.
- La etapa \$project se define con la siguiente sintaxis:  
**{ \$project: { <proyección> } }**
- **<proyección>** es una serie de parejas:
  - **<campo>: <1 o true>** → 'campo' es incluido. Por defecto ningún campo se incluye si no se expresa explícitamente, salvo \_id.
  - **\_id: <0 o false>** → No incluir el campo \_id, que por defecto siempre se incluye.
  - **<campo>: <expresión>** → Añadir un nuevo campo o restablecer el contenido de un campo existente.



# Ejemplo de \$project

- Quedarse únicamente con el campo **age**:

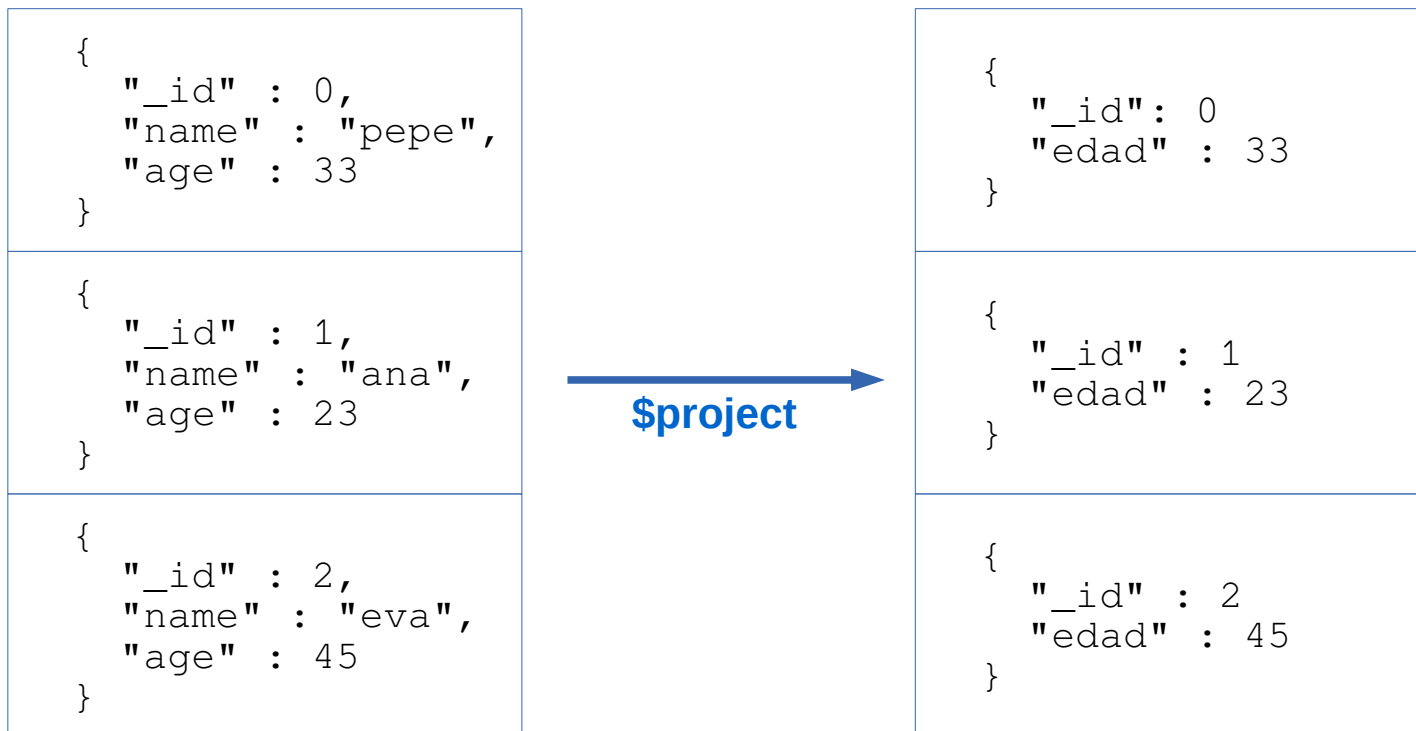
```
db.agg.aggregate([  
  {$project : {_id:0, age:1}}  
])
```



# Ejemplo de \$project

- Renombrar el campo **age** a **edad** y ocultar **name**:

```
db.agg.aggregate([
  {$project : {edad: '$age'}}
])
```

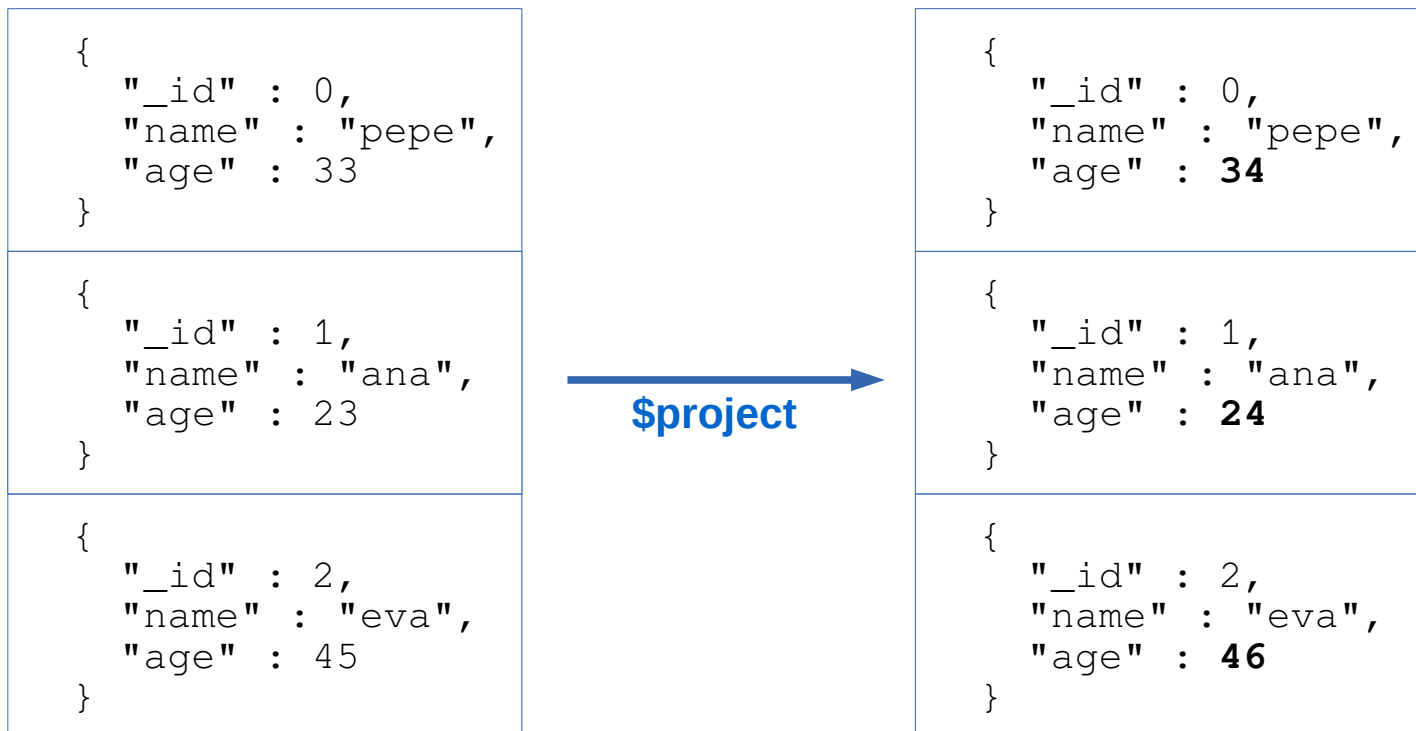


**agg**

# Ejemplo de \$project

- **Aumentar en 1 año el campo **age** de cada documento:**

```
db.agg.aggregate([  
  { $project : { age: { $add: [ '$age', 1 ] }, name: 1 } }  
])
```



**agg**

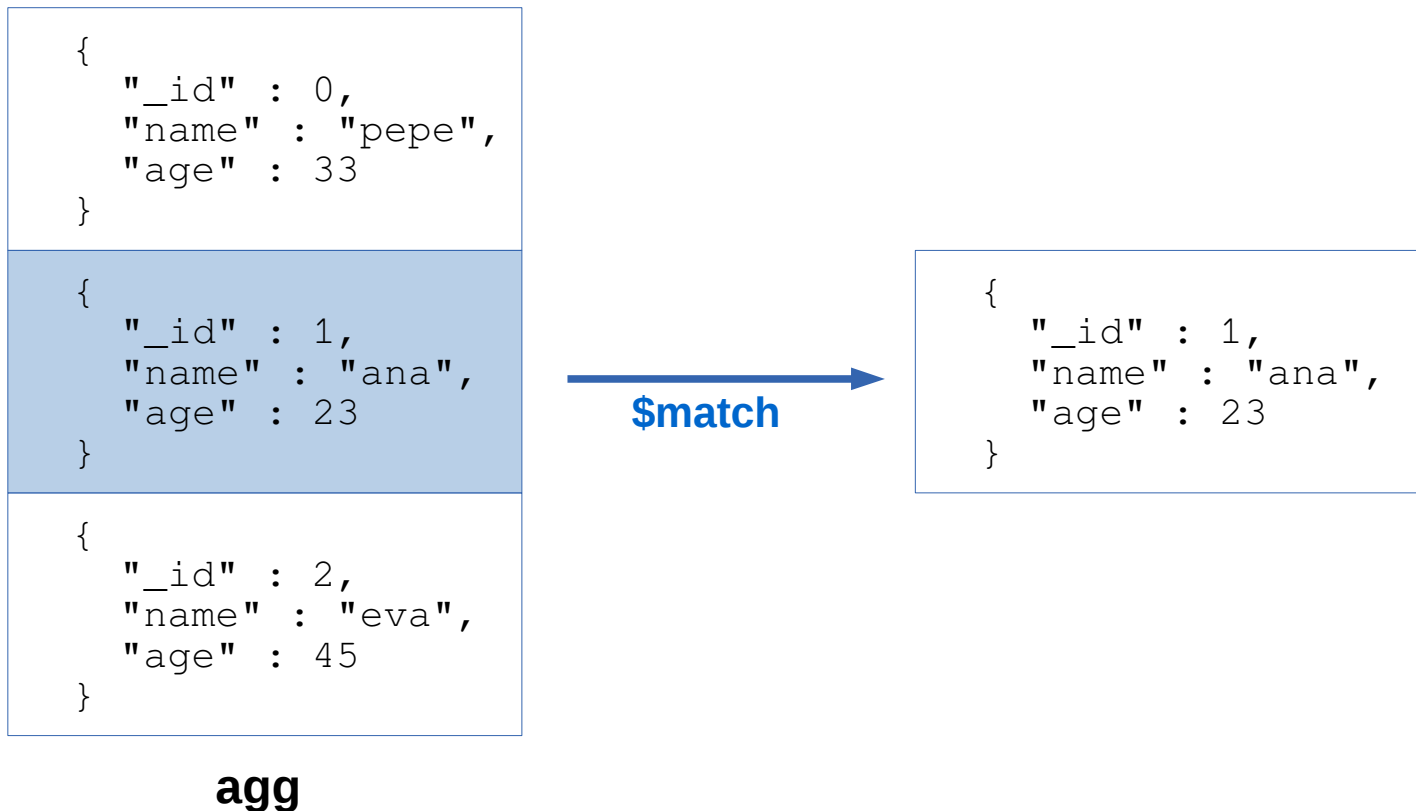
# \$match

- La etapa **\$match** filtra los documentos, dejando pasar únicamente aquellos que cumplen una cierta **propiedad**.
- La sintaxis de la etapa \$match es:  
**{ \$match: { <propiedad> } }**
- **<propiedad>** es una condición representada de la misma manera que las usadas en find().

# Ejemplo de \$match

- Filtrar los documentos cuya **edad** es **23** :

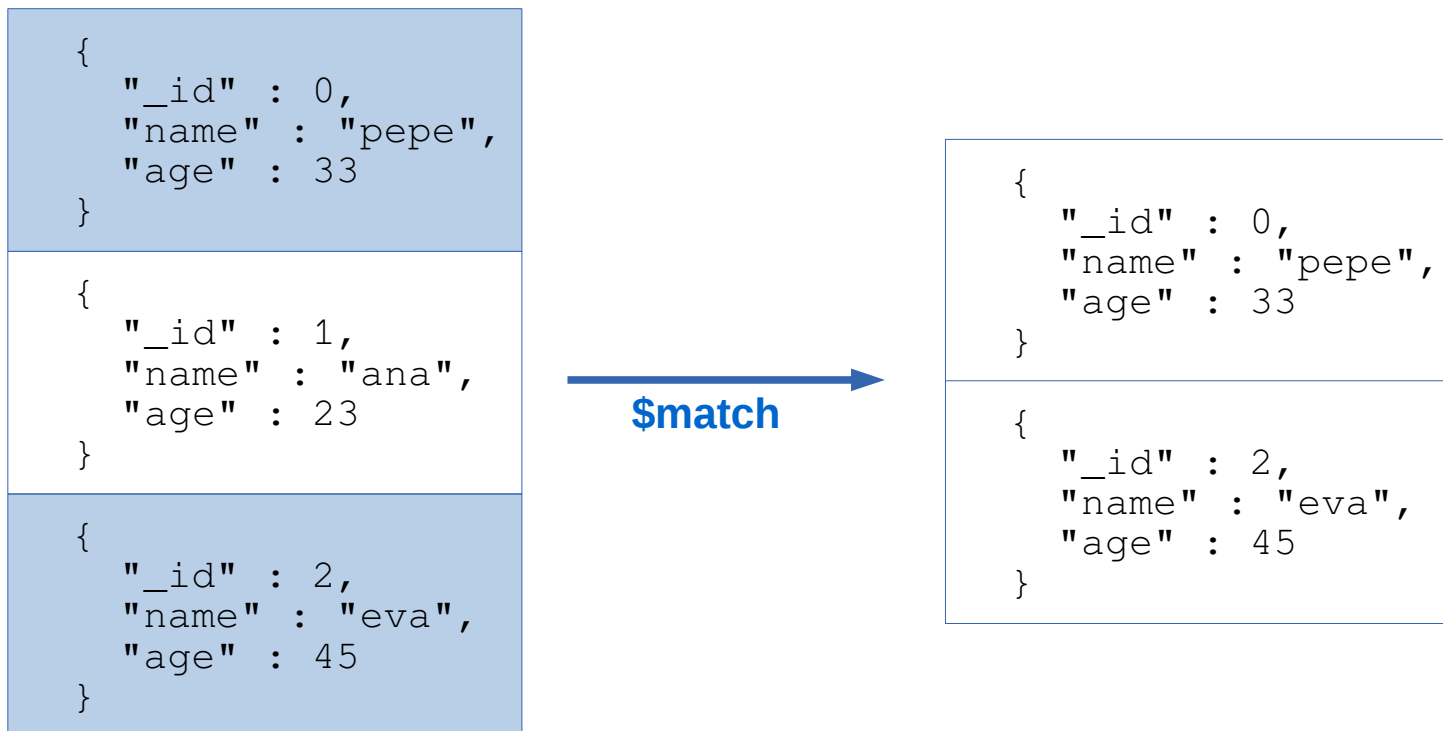
```
db.agg.aggregate([  
  {$match : {age:23}}  
])
```



# Ejemplo de \$match

- Filtrar los documentos con **edad superior a 28 años**:

```
db.agg.aggregate([  
  {$match : {age:{$gt:28}} }  
])
```

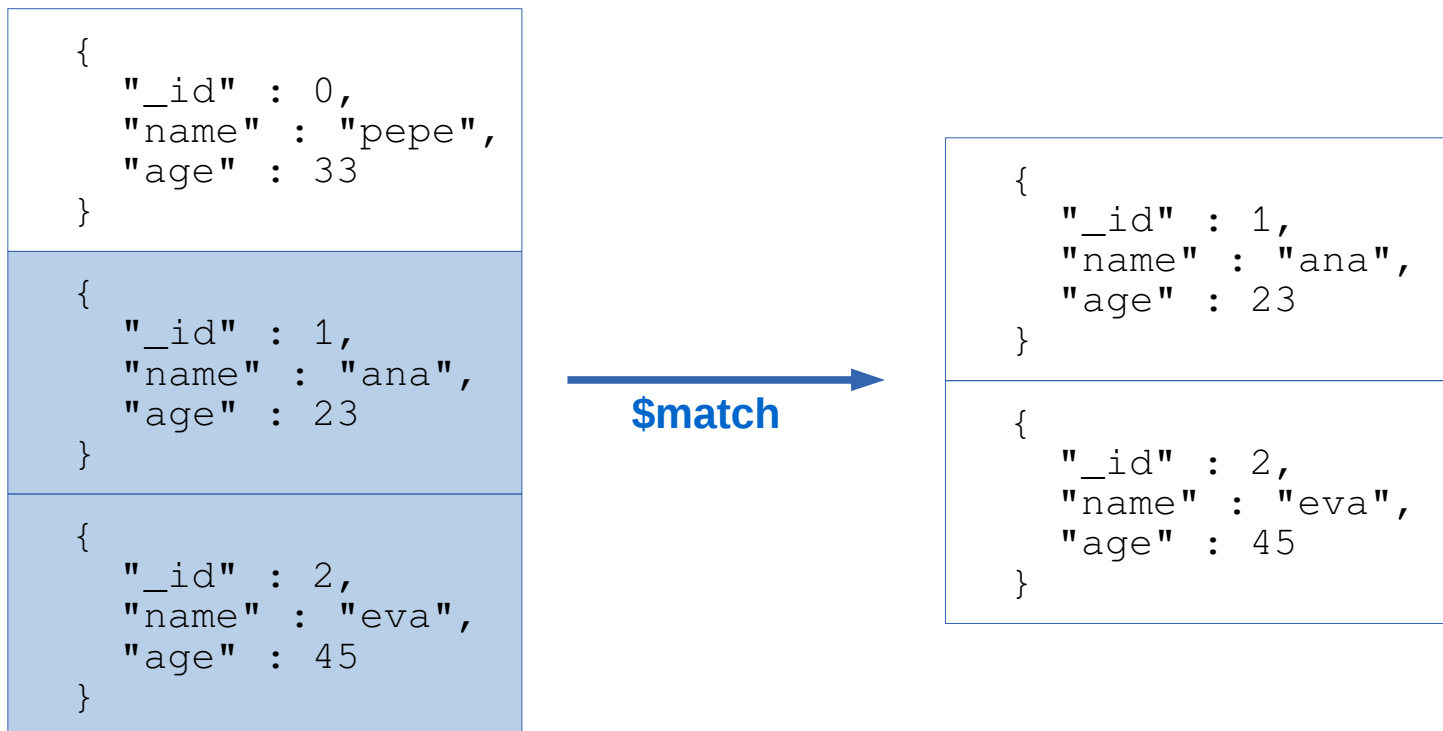


agg

# Ejemplo de \$match

- Filtrar los documentos que tienen **edad inferior a 30** años o se llaman **"eva"**:

```
db.agg.aggregate([  
  {$match: {$or: [{age:{$lt:30}}, {name:"eva"}] }}  
])
```



agg

# Optimizar con \$match

- La etapa \$match **reduce** el número de documentos a aquellos que cumplen la propiedad.
- Por ello es interesante colocar \$match **lo más al principio** de la tubería:
  - Las **etapas siguientes** serán más rápidas pues manejarán **menos documentos**.
  - Al ejecutarse al inicio, es más posible que \$match saque provecho de los **índices**.



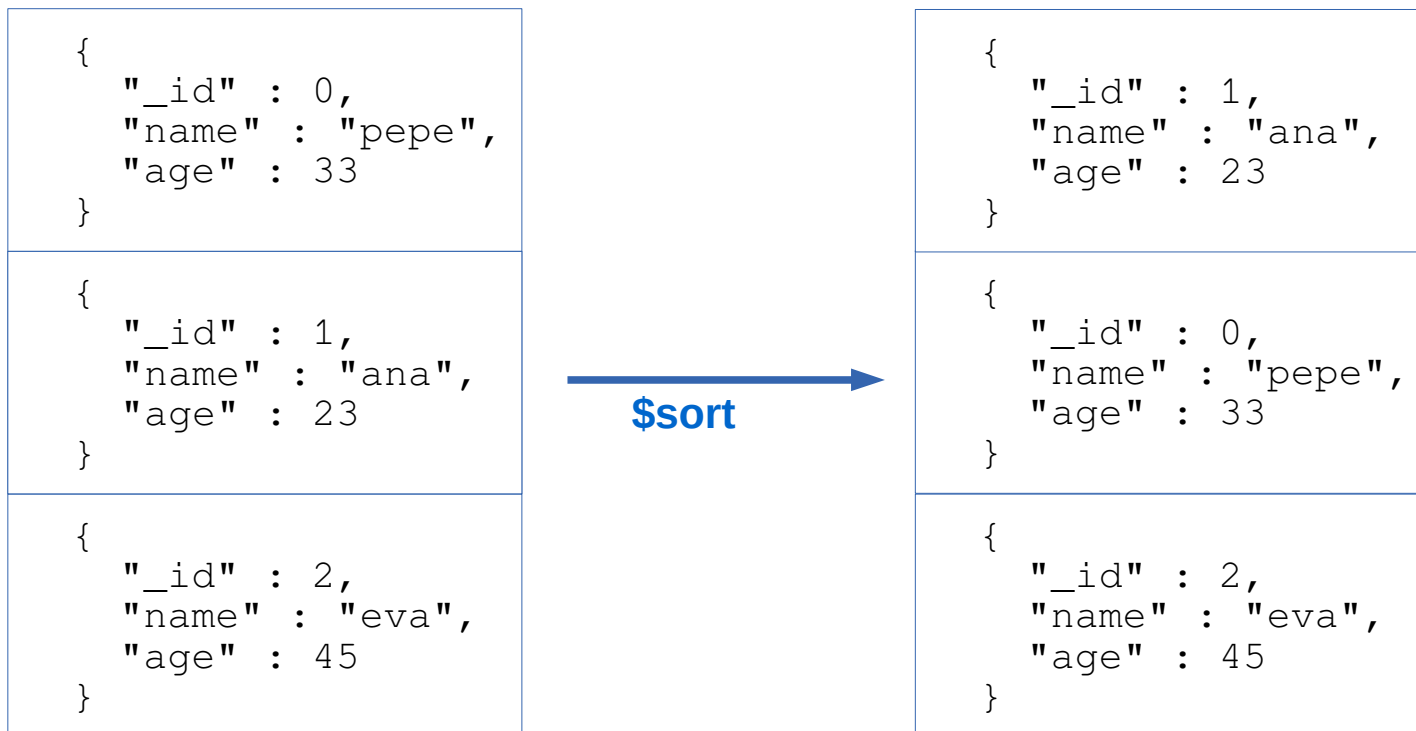
# \$sort

- **Ordena** los documentos recibidos, de acuerdo al orden pasado como parámetro.
- La sintaxis de la etapa \$sort es:  
**{ \$sort: <orden> }**
- **<orden>** es un documento con información de orden representada de la misma manera que la usadas en sort().

# Ejemplo de \$sort

- Ordenar los documentos por **edad ascendente**:

```
db.agg.aggregate([  
  {$sort : {age:1}}  
])
```

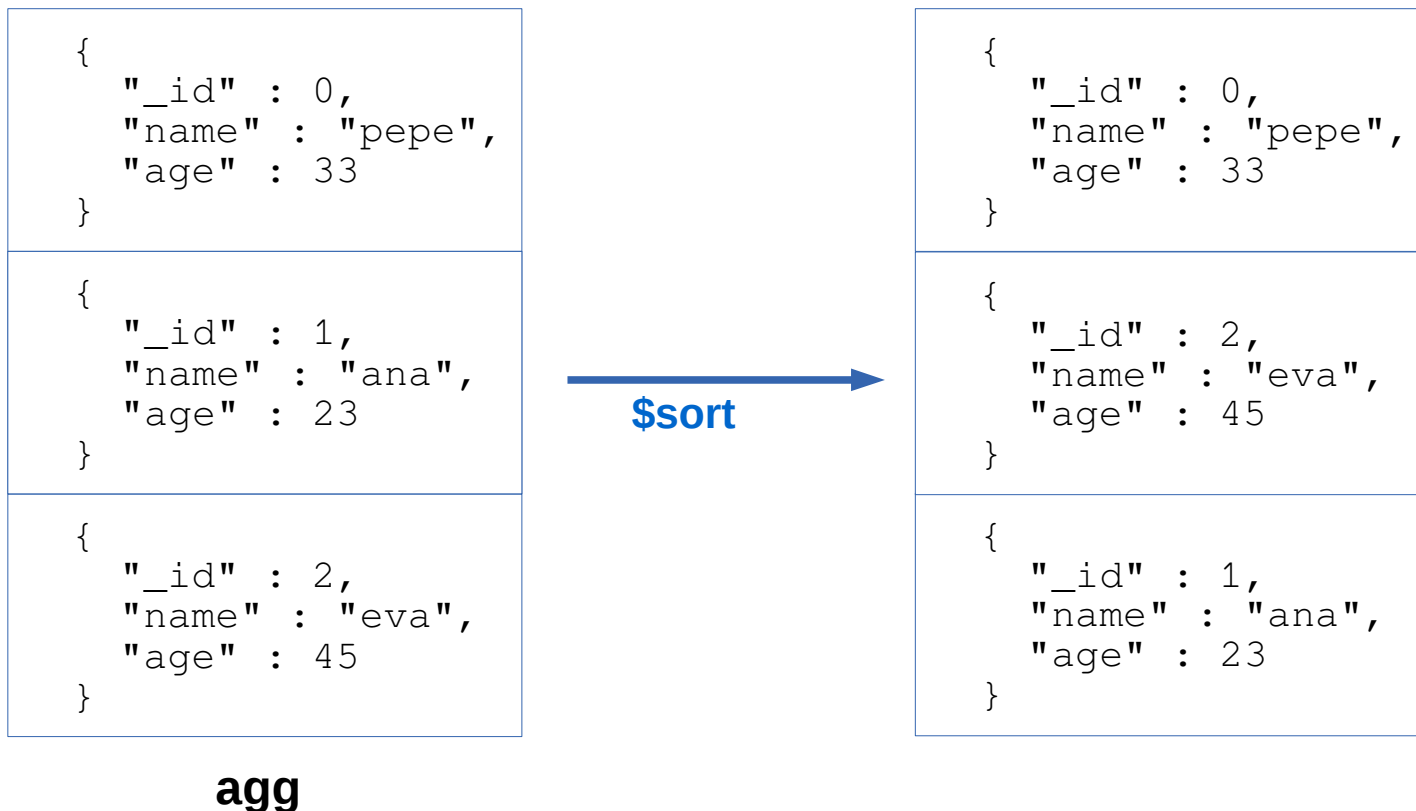


agg

# Ejemplo de \$sort

- Ordenar los documentos por **nombre descendente**, y luego por **\_id ascendente**:

```
db.agg.aggregate([  
  {$sort : {name:-1, _id:1}}  
])
```



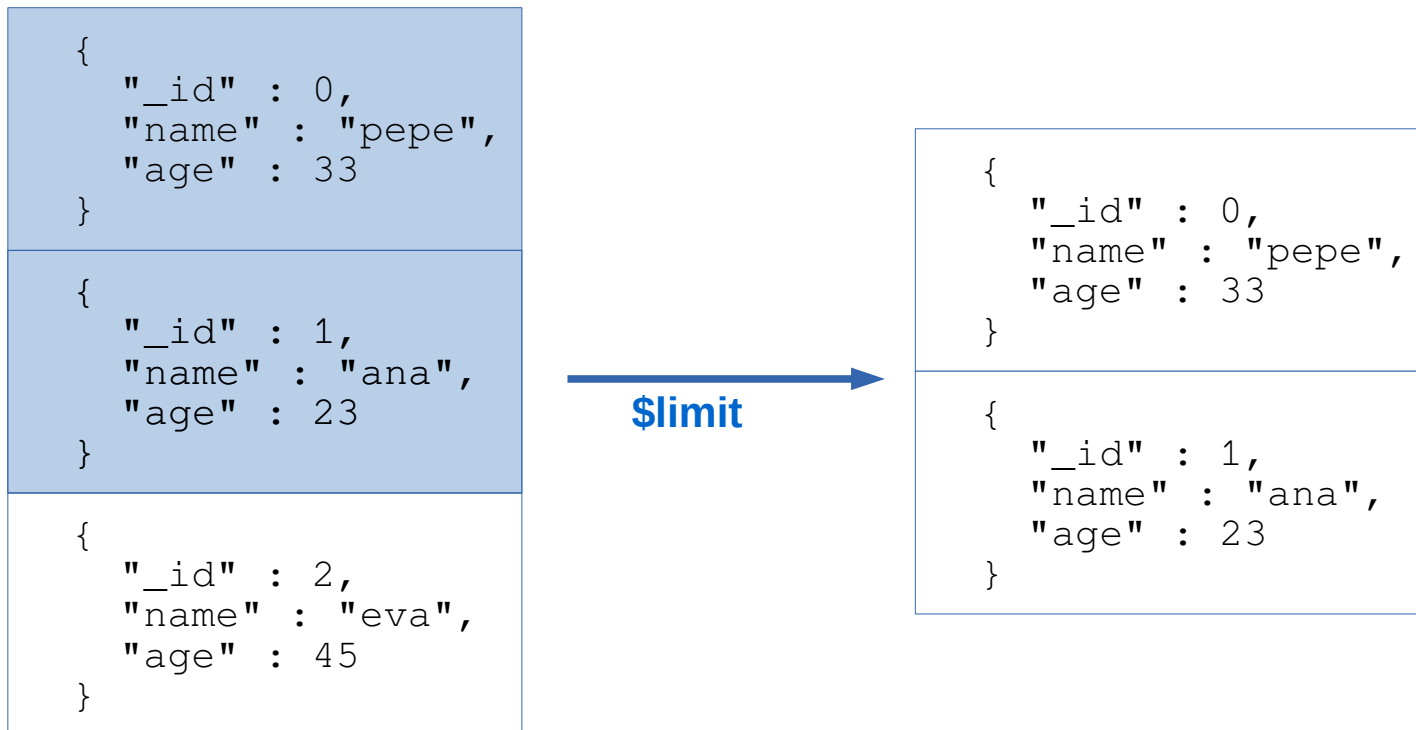
# \$limit

- La etapa \$limit sirve para limitar a un número fijo la cantidad de documentos que pasan a la siguiente etapa.
- La sintaxis de la etapa \$limit es:  
**{ \$limit: <entero positivo> }**
- **<entero positivo>** es cualquier número entero mayor que 0.

# Ejemplo de \$limit

- Limitar la salida a **2 documentos**:

```
db.agg.aggregate([  
  {$limit : 2}  
])
```



**agg**

# \$limit y \$sort

- \$limit devuelve los **primeros 'n'** documentos de entrada, respetando el orden que éstos tienen.
- Si se encadena una etapa **\$sort** y justo después una etapa **\$limit**, MongoDB **optimizará** la etapa \$sort:
  - Durante la ordenación (\$sort) MongoDB únicamente almacenará **'n' resultados** en memoria.

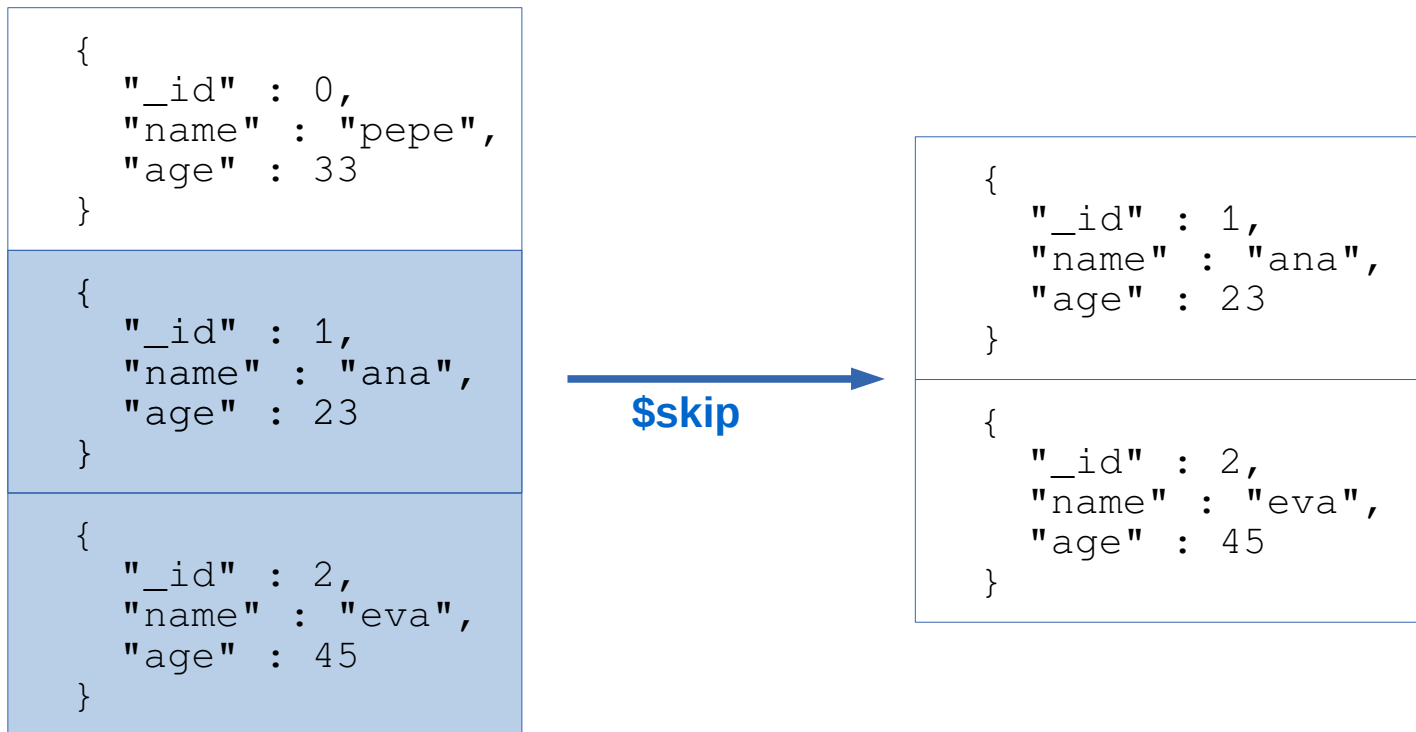
# \$skip

- La etapa \$skip **descarta** los primeros '**n**' **documentos** de entrada y devuelve el resto.
- La sintaxis de la etapa es:  
**{ \$skip: <entero positivo> }**
- **<entero positivo>** es cualquier número entero mayor o igual que 0.

# Ejemplo de \$skip

- **Descartar el primer documento:**

```
db.agg.aggregate([  
  {$skip : 1}  
])
```



**agg**



# \$unwind

- La etapa \$unwind **despliega** un campo que contiene una lista, generando un **documento** por **cada valor de la lista**.
- La sintaxis de la etapa es:  
**{ \$unwind: <campo> }**
- **<campo>** es el nombre del campo que queremos usar para desplegar.

# \$unwind

- Dado un documento:

```
{
  _id : 0,
  state : "Spain",
  cities : ["Cádiz",
            "Valencia",
            "Sevilla"]
}
```

- Al desplegar el documento usando el campo **cities** generaríamos **3 documentos**, uno por cada valor de la lista **cities**:

```
{
  _id : 0,
  state: "Spain",
  cities: "Cádiz"
}
```

```
{
  _id : 0,
  state : "Spain",
  cities : "Valencia"
}
```

```
{
  _id : 0,
  state : "Spain",
  cities : "Sevilla"
}
```

# Ejemplo de \$unwind

- Desplegar por el campo **nums**:

```
db.agg.aggregate([  
  {$unwind: "$nums"}  
])
```

<pre>{   "_id" : 0,   "name" : "ana",   "nums" : [1,4,6] }</pre>
<pre>{   "_id" : 1,   "name" : "lol",   "nums" : [5,0] }</pre>

**agg**

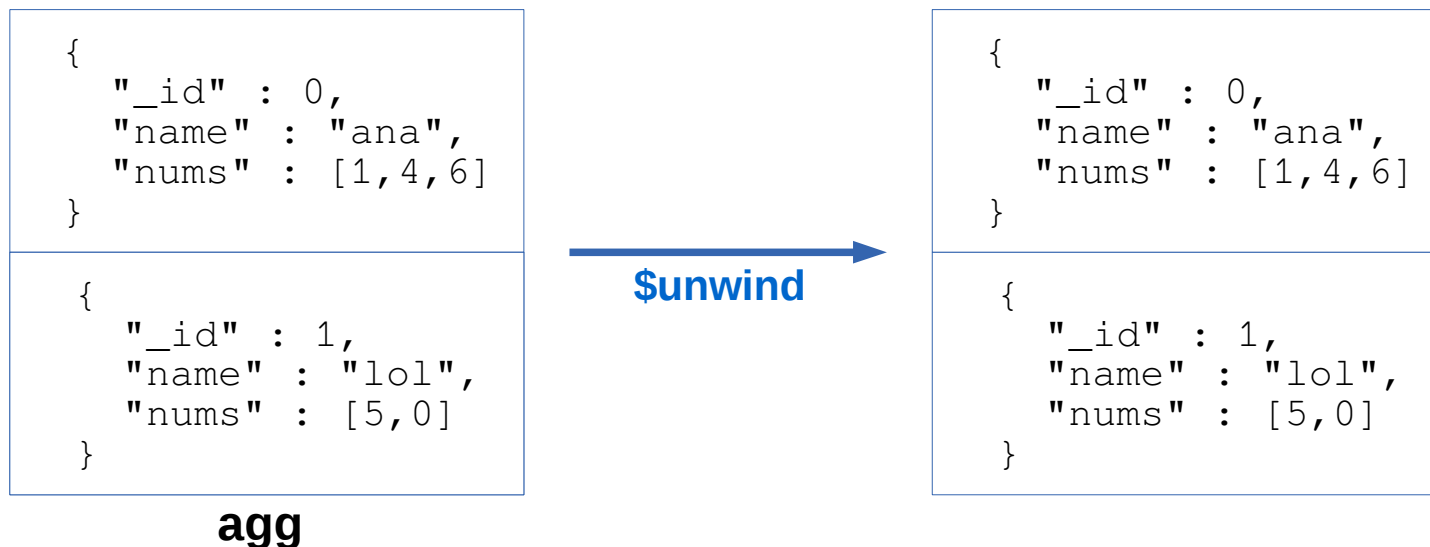
**\$unwind**

<pre>{   "_id" : 0,   "name" : "ana",   "nums" : 1 }</pre>
<pre>{   "_id" : 0,   "name" : "ana",   "nums" : 4 }</pre>
<pre>{   "_id" : 0,   "name" : "ana",   "nums" : 6 }</pre>
<pre>{   "_id" : 1,   "name" : "lol",   "nums" : 5 }</pre>
<pre>{   "_id" : 1,   "name" : "lol",   "nums" : 0 }</pre>

# Ejemplo de \$unwind

- Si en algún documento el **campo** escogido para el desplegado **no contiene una lista**, se tratará como una **lista unitaria**. No lanzará **ningún error**.

```
db.agg.aggregate([  
  { $unwind: "$name" }  
])
```



# Ejemplo de \$unwind

- Si en algún documento el **campo** escogido para el desplegado **no existe**, ese documento se **ignorar**á pero no se lanzará **ningún error**.

```
db.agg.aggregate([  
  {$unwind: "$years" }  
])
```

```
{  
  "_id" : 0,  
  "name" : "ana",  
  "nums" : [1,4,6]  
}  
  
{  
  "_id" : 2,  
  "name" : "pep",  
  "years" : [1,2,4]  
}
```

**agg**

**\$unwind**

```
{  
  "_id" : 2,  
  "name" : "pep",  
  "years" : 1  
}  
  
{  
  "_id" : 2,  
  "name" : "pep",  
  "years" : 2  
}  
  
{  
  "_id" : 2,  
  "name" : "pep",  
  "years" : 4  
}
```

# Ejemplo de \$unwind

- El **campo** escogido para desplegar puede estar **anidado** dentro de otro campo.

```
db.agg.aggregate([  
  {$unwind: "$likes.nums"}  
])
```

agg

```
{  
  "_id" : 0,  
  "name" : "ana",  
  "likes" : {  
    "tot" : 1,  
    "nums" : [1,2]  
  }  
}  
  
{  
  "_id" : 1,  
  "name" : "eva",  
  "likes" : {  
    "tot" : 8,  
    "nums" : [7]  
  }  
}
```

**\$unwind**

```
{  
  "_id" : 0,  
  "name" : "ana",  
  "likes" : {  
    "tot" : 1,  
    "nums" : 1  
  }  
}  
  
{  
  "_id" : 0,  
  "name" : "ana",  
  "likes" : {  
    "tot" : 1,  
    "nums" : 2  
  }  
}  
  
{  
  "_id" : 1,  
  "name" : "eva",  
  "likes" : {  
    "tot" : 8,  
    "nums" : 7  
  }  
}
```

# \$group

- La etapa \$group **agrupa documentos** con el mismo valor en un campo (**clave**) en un solo documento.
- Este documento generado tendrá campos que **combinan** los datos de aquellos que tenían la misma clave:
  - Edad media de los usuario de España.
  - Puntuación máxima en un examen.
  - Lista de todos los usuarios de Francia.

# \$group

- La etapa \$group tiene la sintaxis:  

```
{ $group: { _id: <clave>,  
  <campo1>: { <acumulador1> : <expresion1> }, ... }  
}
```
- **<clave>** es el campo o combinación de campos que se utilizará como clave.
- **<campoN>** es el nombre del campo que almacenará el valor combinado.
- **<acumuladorN>** es el operador acumulador (\$sum, \$avg, \$max, etc.) que se usará para agregar los valores de los documentos.
- **<expresionN>** es la expresión que genera los valores que se acumularán por cada documento. Puede ser el nombre de un campo , una constante, una operación aritmética, de listas, etc.



# Ejemplo de \$group

- Obtener la **edad máxima** de los estudiantes por cada **Universidad**:

```
db.agg.aggregate([
  { $group: { _id: "$edu",
              max_age: { $max: "$age" } } }
])
```

{ "_id" : 0, "name" : "pep", "age" : 33, "edu" : "UPM" }
{ "_id" : 1, "name" : "ana", "age" : 23, "edu" : "UCM" }
{ "_id" : 2, "name" : "eva", "age" : 45, "edu" : "UCM" }

agg

→  
\$group

{ "_id" : "UPM", "max_age" : 33 }
{ "_id" : "UCM", "max_age" : 45 }

# Ejemplo de \$group

- Calcular la **edad media** y el **número de alumnos** por **Universidad**:

```
db.agg.aggregate([  
  $group: {_id: "$edu",  
    avg_age: {$avg: "$age"}, count: {$sum: 1} }  
])
```

<pre>{   "_id" : 0,   "name" : "pep",   "age" : 33,   "edu" : "UPM" }</pre>
<pre>{   "_id" : 1,   "name" : "ana",   "age" : 23,   "edu" : "UCM" }</pre>
<pre>{   "_id" : 2,   "name" : "eva",   "age" : 45,   "edu" : "UCM" }</pre>

agg

→  
\$group

<pre>{   "_id" : "UPM",   "avg_age" : 33,   "count" : 1 }</pre>
<pre>{   "_id" : "UCM",   "avg_age" : 34,   "count" : 2 }</pre>

# Ejemplo de \$group

- Obtener los nombres de los **alumnos** de cada **Universidad** en un único documento:

```
db.agg.aggregate([  
  {$group: {_id: "$edu",  
            names: {$push: "$name"} }}  
])
```

{ "_id" : 0, "name" : "pep", "age" : 33, "edu" : "UPM" }
{ "_id" : 1, "name" : "ana", "age" : 23, "edu" : "UCM" }
{ "_id" : 2, "name" : "eva", "age" : 45, "edu" : "UCM" }

agg

→  
\$group

{ "_id" : "UPM", "names" : ["pep"] }
{ "_id" : "UCM", "names" : ["ana", "eva"] }

# \$lookup

- La etapa \$lookup realiza una combinación externa izquierda (***left outer join***) de los documentos de entrada con otra colección.
- La combinación se realiza comparando dos campos de los documentos que pueden tener un **nombre diferente**: uno de la colección **local** y otro de la **externa**.
- El documento de la colección se **extiende** con una **lista** conteniendo los documentos de la otra colección que tienen el mismo valor en el campo.

# \$lookup

- La sintaxis de la etapa \$lookup es:

```
{ $lookup: {  
  from: <colección externa a combinar>,  
  localField: <nombre del campo de los docs.  
    de la colección local a comparar>,  
  foreignField: <nombre del campo de los docs. De  
    la colección externa a comparar>,  
  as: <nombre del campo generado que almacenará la  
    lista>  
}}
```

# Ejemplo de \$lookup

- Combinar usuarios con sus pedidos:

```
db.users.aggregate([  
  $lookup: { from : "orders",  
             localField : "dni",  
             foreignField : "user",  
             as : "pedidos" }  
])
```

```
{  
  "_id" : 0,  
  "dni" : "125G",  
  "name" : "ana"  
}
```

```
{  
  "_id" : 1,  
  "dni" : "845K",  
  "name" : "pep"  
}
```

users

```
{  
  "_id" : 0,  
  "user" : "125G",  
  "total" : 87651  
}
```

```
{  
  "_id" : 1,  
  "user" : "125G",  
  "total" : 4562  
}
```

orders

➔  
\$lookup

```
{  
  "_id" : 0,  
  "dni" : "125G",  
  "name" : "ana",  
  "pedidos": [  
    { "_id": 0,  
      "user" : "125G",  
      "total" : 87651 },  
    { "_id" : 1,  
      "user" : "125G",  
      "total" : 4562 } ]  
}
```

```
{  
  "_id" : 1,  
  "dni" : "845K",  
  "name" : "pep",  
  "pedidos" : []  
}
```

# \$lookup

- El nuevo campo añadido por \$lookup contiene una lista con los documentos **completos** que encajan.
- Este nuevo campo puede ser una **lista vacía** si en la colección externa no hay ningún documento que encaje.
- La colección externa debe pertenecer a la **misma base de datos**.

# \$out

- La etapa \$out **almacena** los documentos en un **colección**.
- La colección de salida se **crea** si no existe, y si existe se **reemplaza completamente**.
- La sintaxis de \$out es:  
**{ \$out: <colección de salida> }**



# \$out

- La etapa \$out fallará si existen documentos con el campo **\_id** duplicado.
- Si los documentos carecen de \_id, éste campo será **generado automáticamente**.
- La etapa \$out debe ser la **última de la tubería**.
- Ejemplo: volcar los resultados en la colección **summary**:

```
> db.users.aggregate([  
    ..., // Etapas anteriores  
    { $out : "summary" }  
])
```

**Conectar tuberías**

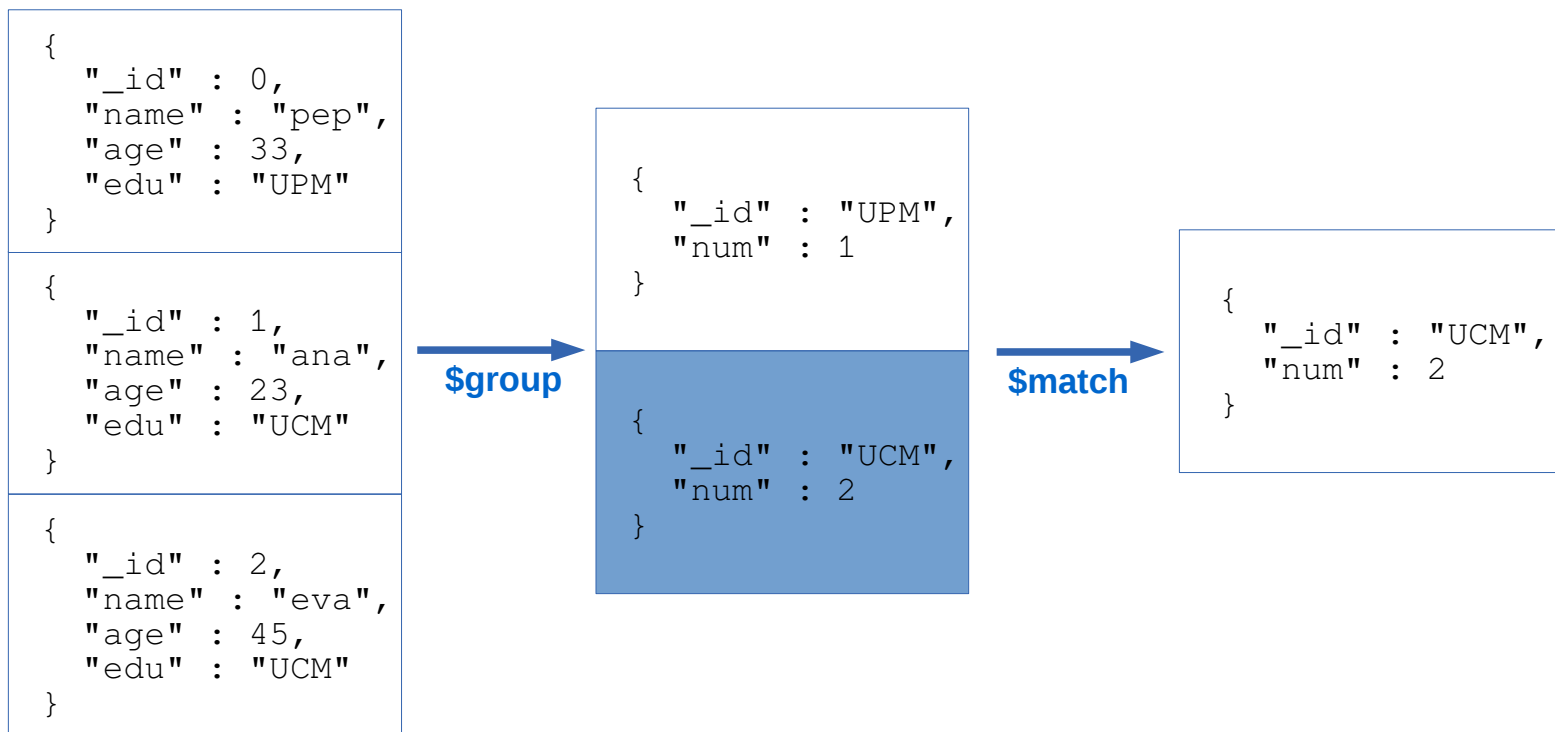
# Conectando tuberías

- Hasta ahora hemos visto ejemplos de tuberías de una sola etapa, que toman la entrada directamente de una colección.
- La potencia del *aggregation pipeline* consiste en su **combinar etapas**.
- Para ello solo es necesario definir las etapas en el orden deseado.

# Ejemplos de tuberías

- Universidades con 2 o más alumnos:

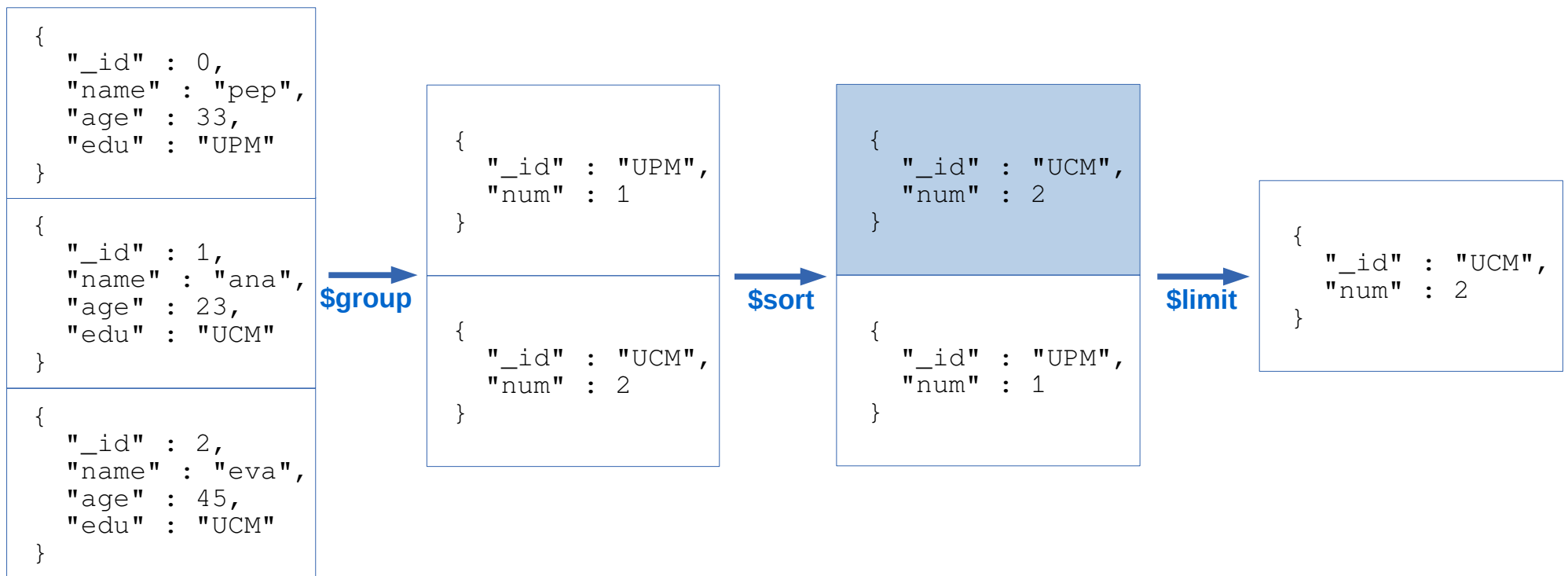
```
db.students.aggregate([  
  { $group : { "_id": "$edu", "num": { $sum: 1 } } },  
  { $match : { "num": { $gte: 2 } } }  
])
```



# Ejemplos de tuberías

- Universidad que tiene más alumnos:

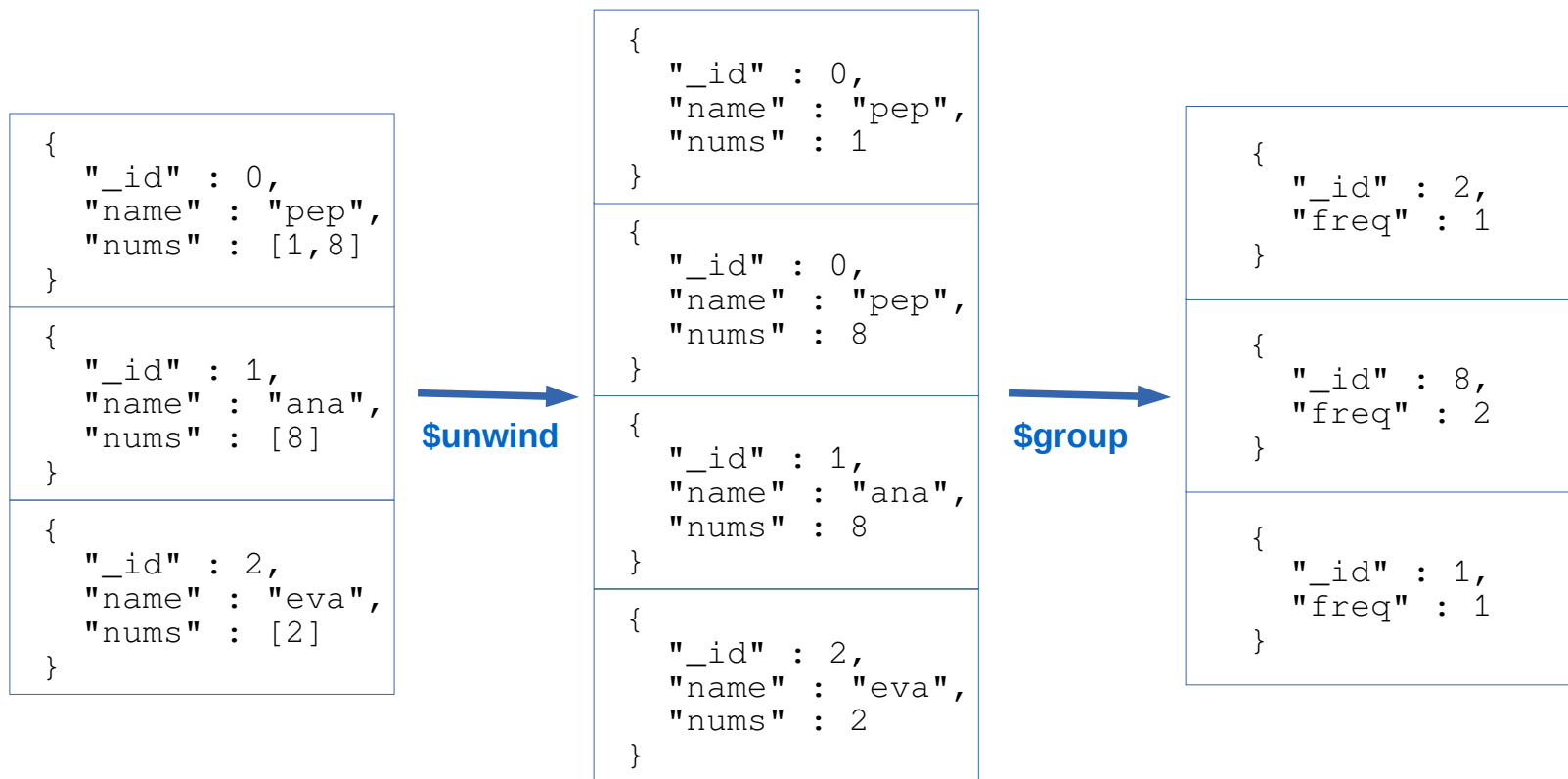
```
db.students.aggregate([  
  { $group : { "_id": "$edu", "num": { $sum: 1 } } },  
  { $sort : { "num" : -1 } },  
  { $limit : 1 },  
])
```



# Ejemplos de tuberías

- Frecuencia de cada número:

```
db.students.aggregate([
  {$unwind: "$nums"},
  {$group: { "_id": "$nums", "freq": {"$sum": 1} }}
])
```



# Ejercicios

# Ejercicio 1

- **Nombre** de la universidad con mayor edad media, dentro del campo "univ".

```
{
  "_id" : 0,
  "name" : "pep",
  "age" : 33,
  "edu" : "UPM"
}

{
  "_id" : 1,
  "name" : "ana",
  "age" : 23,
  "edu" : "UCM"
}

{
  "_id" : 2,
  "name" : "eva",
  "age" : 45,
  "edu" : "UCM"
}
```

**students**



```
{
  "univ" : "UCM"
}
```

**Resultado  
esperado**



# Ejercicio 2

- Listado de **números más frecuentes** y **lista de usuarios, ordenados** por:
  1. número de apariciones, descendente
  2. número, descendente

```
{
  "_id" : 0,
  "name" : "pep",
  "nums" : [1, 8]
}

{
  "_id" : 1,
  "name" : "ana",
  "nums" : [8]
}

{
  "_id" : 2,
  "name" : "eva",
  "nums" : [2]
}
```

**students**



```
{
  "_id" : 8,
  "freq" : 2
  "users" : [0, 1]
}

{
  "_id" : 2,
  "freq" : 1
  "users" : [2]
}

{
  "_id" : 1,
  "freq" : 1
  "users" : [0]
}
```

**Resultado  
esperado**

# Ejercicio 3

- **Usuarios** junto con el **número de pedidos** que han realizado (sin usuarios con 0 pedidos).

```
{
  "_id" : 0,
  "name" : "pep",
  "age" : 33
}
{
  "_id" : 1,
  "name" : "ana",
  "age" : 23
}
{
  "_id" : 2,
  "name" : "eva",
  "age" : 45
}
```

**users**

```
{
  "_id" : 0,
  "user" : 0,
  "total" : 88
}
{
  "_id" : 1,
  "user" : 0,
  "total" : 128
}
{
  "_id" : 2,
  "user" : 2,
  "total" : 1254
}
{
  "_id" : 3,
  "user" : 0,
  "total" : 98
}
```

**orders**



```
{
  "_id" : 0,
  "name" : "pep",
  "age" : 33,
  "n_orders" : 3
}
{
  "_id" : 2,
  "name" : "eva",
  "age" : 45,
  "n_orders" : 1
}
```

**Resultado  
esperado**

# Ejercicio 4

- **Usuarios** junto con **gasto total** de sus compras, sólo si han gastado más de 1000€. Ordenados de manera descendente por gasto total.

```
{
  "_id" : 0,
  "name" : "pep",
  "age" : 33
}
{
  "_id" : 1,
  "name" : "ana",
  "age" : 23
}
{
  "_id" : 2,
  "name" : "eva",
  "age" : 45
}
```

**users**

```
{
  "_id" : 0,
  "user" : 0,
  "total" : 88
}
{
  "_id" : 1,
  "user" : 0,
  "total" : 128
}
{
  "_id" : 2,
  "user" : 2,
  "total" : 1254
}
{
  "_id" : 3,
  "user" : 0,
  "total" : 98
}
```

**orders**



```
{
  "_id" : 2,
  "name" : "eva",
  "age" : 45,
  "total" : 1254
}
```

**Resultado  
esperado**

# **Tuberías de agregación en pymongo**

# aggregate()

- Los objetos **Collection** de pymongo tienen un método **aggregate** (ver detalles en referencias).
- El método aggregate() recibe una **lista Python de etapas** representadas como diccionarios.
- La sintaxis de cada etapa es **exactamente igual** a la usada en la **consola** de MongoDB.

# aggregate()

```
from pymongo import MongoClient
```

```
mongoclient = MongoClient()
```

```
db = mongoclient[ 'giw' ]
```

```
c = db[ 'users' ]
```

```
c.insert_one( { 'name': 'ana', '_id':0, 'edad':24 } )
```

```
c.insert_one( { 'name': 'eva', '_id':1, 'edad':28 } )
```

```
c.insert_one( { 'name': 'pep', '_id':2, 'edad':34 } )
```

```
r = c.aggregate( [
    { '$match': { 'edad': { '$gt': 25 } } },
    { '$sort': { 'edad': -1 } }
] )
```

```
for e in r:
```

```
    # procesar e
```

# Referencias

# Documentación de MongoDB

- Todas las etapas del ***aggregation pipeline***:  
<https://docs.mongodb.com/manual/reference/operator/aggregation/#aggregation-pipeline-operator-reference>
- Etapa **\$project**:  
[https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe.\\_S\\_project](https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project)
- Etapa **\$match**:  
[https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe.\\_S\\_match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match)



# Documentación de MongoDB

- Etapa **\$sort**:  
[https://docs.mongodb.com/manual/reference/operator/aggregation/sort/#pipe.\\_S\\_sort](https://docs.mongodb.com/manual/reference/operator/aggregation/sort/#pipe._S_sort)
- Etapa **\$limit**:  
[https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe.\\_S\\_limit](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit)
- Etapa **\$skip**:  
[https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe.\\_S\\_skip](https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe._S_skip)

# Documentación de MongoDB

- Etapa **\$unwind**:

[https://docs.mongodb.com/manual/reference/operator/aggregation/unwind/#pipe.\\_S\\_unwind](https://docs.mongodb.com/manual/reference/operator/aggregation/unwind/#pipe._S_unwind)

- Etapa **\$group**:

[https://docs.mongodb.com/manual/reference/operator/aggregation/group/#pipe.\\_S\\_group](https://docs.mongodb.com/manual/reference/operator/aggregation/group/#pipe._S_group)

- Etapa **\$lookup**:

[https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#pipe.\\_S\\_lookup](https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#pipe._S_lookup)

# Documentación de MongoDB

- Etapa **\$out**:

[https://docs.mongodb.com/manual/reference/operator/aggregation/out/#pipe.\\_S\\_out](https://docs.mongodb.com/manual/reference/operator/aggregation/out/#pipe._S_out)

- Método **aggregate()** de pymongo:

<https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.aggregate>