



mongoDB

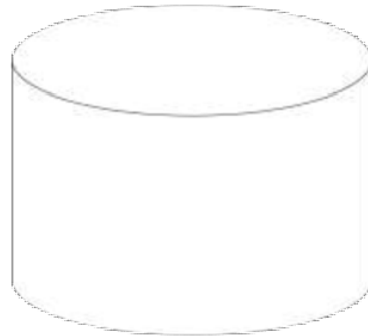
# AGREGATION FRAMEWORK

Enrique Barra

# Analytics in MongoDB?



**Create**  
**Read**  
**Update**  
**Delete**



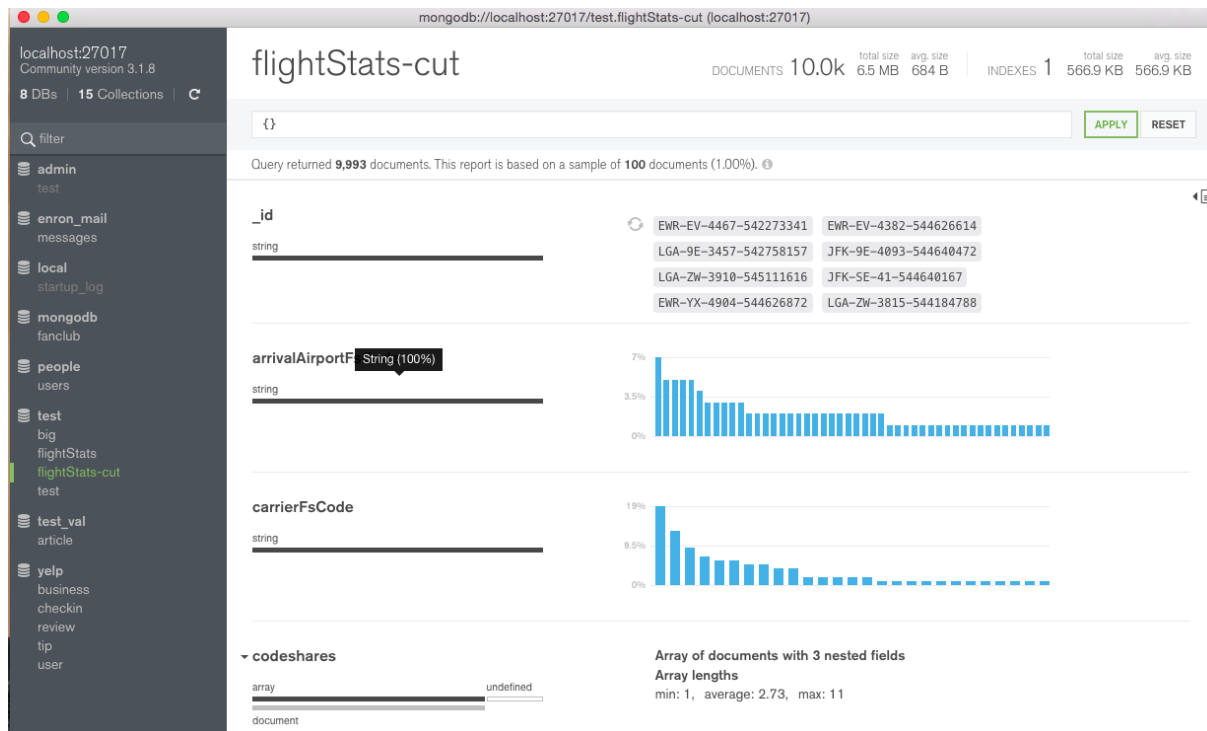
**Group**  
**Count**  
**Derive Values**  
**Filter**  
**Average**  
**Sort**

# QUE ES EL AGGREGATION FRAMEWORK

- Las agregaciones son operaciones que procesan entradas y devuelven resultados **calculados**.
- Estas operaciones agrupan valores de múltiples documentos juntos, y pueden realizar una gran variedad de operaciones sobre los datos para devolver un resultado simple.
- Tres tipos de agregación:
  1. **The aggregation pipeline**
    - <https://docs.mongodb.org/manual/core/aggregation-pipeline/>
  2. **The map-reduce function**
    - <https://docs.mongodb.org/manual/core/map-reduce/>
  3. **The single purpose aggregation methods and commands**
    - <https://docs.mongodb.org/manual/core/single-purpose-aggregation/>
- Comparativa de los 3 tipos de agregación:  
<https://docs.mongodb.org/manual/reference/aggregation-commands-comparison/>
- Más info: <https://docs.mongodb.org/manual/aggregation/>
- Ejemplos: <https://docs.mongodb.org/v3.0/applications/aggregation/>
- Video: <https://www.youtube.com/watch?v=9HJxi7Q7YJA>

# MONGODB COMPASS

- <https://www.mongodb.com/products/compass>
- Es una GUI para MongoDB
- Funcionalidades:
  - explorar los datos visualmente,
  - ejecutar queries,
  - toda la funcionalidad CRUD



# NOVEDAD EN COMPASS – CREADOR DE PIPELINES

test.flightStats

DOCUMENTS 123TOTAL SIZE 56.4KBAVG. SIZE 469BINDEXES 3TOTAL SIZE 116.0KBAVG. SIZE 38.7KB

DocumentsAggregationsSchemaExplain PlanIndexesValidation

Flight Stats PipelineSAVE PIPELINE...COMMENT MODESAMPLE MODEAUTO PREVIEWUnsaved changes

123 Documents in the Collection

Select an operator to construct expressions used in the aggregation pipeline stages. [Learn more](#)

Preview of Documents in the Collection

```
{
  "_id": "EWR-1I-330-543184347",
  "arrivalAirportFsCode": "CMI",
  "cancelled": 0,
  "carrierFsCode": "1I",
  "departureAirportFsCode": "EWR",
  "departureDate": {
    "$date": "2015-05-15T16:31:00.000Z"
  },
  "departureLocal": {
    "$date": "2015-05-15T16:31:00.000Z"
  },
  "delays": {
    "arrivalRunwayDelayMinutes": 3
  }
}
```

```
{
  "_id": "EWR-1I-382-544589251",
  "arrivalAirportFsCode": "PHL",
  "cancelled": 0,
  "carrierFsCode": "1I",
  "departureAirportFsCode": "EWR",
  "departureDate": {
    "$date": "2015-05-15T16:31:00.000Z"
  },
  "departureLocal": {
    "$date": "2015-05-15T16:31:00.000Z"
  },
  "delays": {
    "arrivalRunwayDelayMinutes": 3
  }
}
```

\$match

Output after \$match stage (Sample of 6 documents)

```
1 /*
2  * query - The query in MQL.
3  */
4 {
5   carrierFsCode: "1I"
6 }
```

```
{
  "_id": "EWR-1I-330-543184347",
  "arrivalAirportFsCode": "CMI",
  "cancelled": 0,
  "carrierFsCode": "1I",
  "departureAirportFsCode": "EWR",
  "departureDate": {
    "$date": "2015-05-15T16:31:00.000Z"
  },
  "departureLocal": {
    "$date": "2015-05-15T16:31:00.000Z"
  },
  "delays": {
    "arrivalRunwayDelayMinutes": 3
  }
}
```

```
{
  "_id": "EWR-1I-382-544589251",
  "arrivalAirportFsCode": "PHL",
  "cancelled": 0,
  "carrierFsCode": "1I",
  "departureAirportFsCode": "EWR",
  "departureDate": {
    "$date": "2015-05-15T16:31:00.000Z"
  },
  "departureLocal": {
    "$date": "2015-05-15T16:31:00.000Z"
  },
  "delays": {
    "arrivalRunwayDelayMinutes": 3
  }
}
```

\$group

Output after \$group stage (Sample of 1 document)

```
1 /*
2  * _id - The id of the group.
3  * field1 - The first field name.
4  */
5 {
6   _id: "$departureAirportFsCode",
7   flights: {
8     $push: "$flightId"
9   }
10 }
```

```
{
  "_id": "EWR",
  "flights": [
    0: 543184347,
    1: 544589251,
    2: 544589200,
    3: 543183182,
    4: 545515483,
    5: 544595864
  ]
}
```

ADD STAGE

A decorative graphic on the left side of the slide. It features several vertical lines of varying shades of green. Overlaid on these lines are several green circles of different sizes. One large circle is positioned near the top, and several smaller circles are arranged below it, some overlapping the vertical lines.

# AGGREGATION PIPELINE

6

# AGGREGATION PIPELINE

- El framework de agregación está diseñado siguiendo el concepto del procesamiento de datos con pipelines (tuberías)
- La entrada de un pipeline es una colección, dentro de la pipeline se produce un procesamiento de los documentos de esa colección y la salida de una pipeline es la entrada de la siguiente
- Se dice que funciona en fases o “stages”
- Cada fase es:
  - **Filtro** que opera como las queries
  - **Transformación** que modifican la forma de salida del documento
  - **Agrupación** por campos
  - **Ordenación** por campos
- Notas:
  - Funciona en particiones
  - Las fases pueden usar índices para mejorar la eficiencia
- Más info sobre fases y operadores:
  - <https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/>
  - <https://docs.mongodb.com/manual/reference/operator/aggregation/>

## **db.collection.aggregate(pipeline, options)**

- **Pipeline:** es un **array** con una secuencia de operaciones o stages (fases)
- **Options:** opciones extra: explain, allowDiskUse, cursor, bypassDocumentValidation, readConcern
- Pipeline operators:
  - **\$match:** Filtrar documentos
  - **\$project:** como projection en el find, obtener sólo lo que queremos de cada documento o transformar los documentos
  - **\$addFields:** similar a project, transforma el doc añadiéndole campos (muy útil combinado con \$convert)  
<https://docs.mongodb.com/manual/reference/operator/aggregation/convert/>
  - **\$group:** agrupar documentos
  - **\$unwind:** expandir arrays y subdocumentos de los documentos
  - **\$sort:** ordenar
  - **\$limit/\$skip:** paginar documentos
  - **\$redact:** restringir documentos
  - **\$geoNear:** ordenar por proximidad
  - **\$lookup:** left outer join con otra colección de la misma bbdd
  - **\$out:** escribir los resultados a una colección
  - **\$let, \$map:** definir variables
  - ...



# FASE \$GROUP Y SUS ACUMULADORES

- <https://docs.mongodb.org/v3.0/reference/operator/aggregation-group>
- La fase \$group **calcula** valores procesando documentos que **comparten una clave**
- Pasa a la fase siguiente un documento **por cada agrupación** que realiza
- El documento de salida tiene un campo `_id` que contiene la clave que se ha usado para la agrupación y puede contener campos calculados con valores de algún acumulador
- Acumuladores:
  - **\$sum**: suma
  - **\$avg**: media
  - **\$first**: primer documento
  - **\$last**: último
  - **\$max**: máximo
  - **\$min**: mínimo
  - **\$push**: devuelve un array con todos los valores procesados por una expresión
  - **\$addToSet**: devuelve un array con todos los valores procesados por una expresión y únicos

# FRAMEWORK DE AGREGACIÓN – EJEMPLO \$GROUP

```
{  
  "_id" : ObjectId("50b1aa983b3d0043b51b2c52"),  
  "name" : "Nexus 7",  
  "category" : "Tablets",  
  "manufacturer" : "Google",  
  "price" : 199  
}
```

- Query que suma todos los productos de cada categoría:

```
db.products.aggregate([  
  { $group: { _id: "$category", num_products: {$sum:1} } }  
])
```

# FRAMEWORK DE AGREGACIÓN – EJEMPLO \$GROUP 2

```
{  
  "_id" : ObjectId("50b1aa983b3d0043b51b2c52"),  
  "name" : "Nexus 7",  
  "category" : "Tablets",  
  "manufacturer" : "Google",  
  "price" : 199  
}
```

- Query que calcula el precio medio de los productos de cada categoría:

```
db.products.aggregate([  
  { $group: { _id: "$category", avg_price: { $avg: "$price" } } }  
])
```

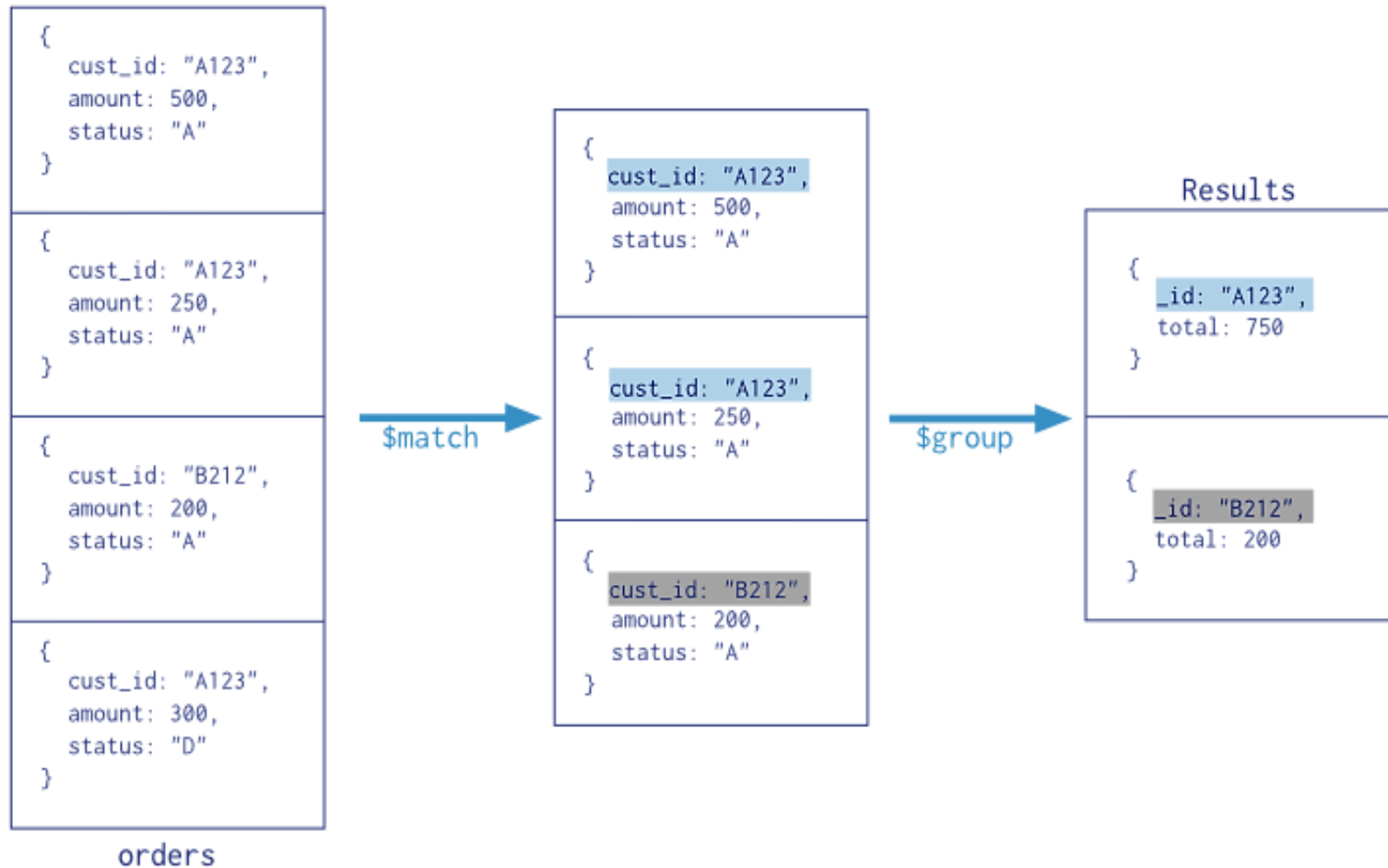
# FASE \$MATCH

- Filtra los documentos y pasa solo los que cumplan determinada condición (como vimos en el operador `.find()` de la shell)
- Muy útil para aplicar **antes** de las otras fases si quiero hacer cálculos solo sobre determinados documentos
- Muy útil para aplicar **después** de las otras fases si quiero devolver sólo algunos documentos que cumplan alguna condición

# AGGREGATION PIPELINE

QUEREMOS EL GASTO TOTAL POR CLIENTE CON STATUS "A"

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



# FASE \$PROJECT

- Pasa los documentos a la fase siguiente transformándolos.
- Se puede pasar solo algunos campos (como el projection de la shell)
- Se pueden pasar nuevos campos calculados
  - Substring (\$substr)
  - Restas (\$subtract)
  - Uppercase (\$toUpper)
  - Condiciones (\$if)
  - Añadir campos (\$addFields)
  - ...

# EJEMPLOS FASE \$PROJECT

- Ejemplo:

```
{  
  _id : "jane",  
  joined : ISODate("2011-03-02"),  
  likes : ["golf", "racquetball"]  
}  
  
{  
  _id : "joe",  
  joined : ISODate("2012-07-02"),  
  likes : ["tennis", "golf", "swimming"]  
}
```

- Queremos los nombres en orden alfabético y en mayúsculas

# NOMBRES EN ORDEN ALFABÉTICO Y EN MAYÚSCULAS

```
db.users.aggregate(  
  [  
    { $project : { name: { $toUpper: "$_id" } , _id:0 } },  
    { $sort : { name : 1 } }  
  ]  
)
```

- La fase \$project:
  - Crea un nuevo campo llamado “name” con el \_id convirtiéndolo a mayúsculas con el operator \$toUpper
  - Suprime el campo \_id (el \_id se pasa por defecto, salvo que se suprima explícitamente)
- La fase \$sort recibe estos documentos y como ya tienen campo name puede ordenarlos por este campo
- Resultado: { "name" : "JANE" }, { "name" : "JILL" }, { "name" : "JOE" }...



# FASE \$UNWIND

- **Deconstruye un campo array** de los documentos de entrada
- Cada documento de salida es el mismo que el de entrada con el valor del array reemplazado por el elemento

- Ejemplo:

```
{
  _id : "jane",
  joined : ISODate("2011-03-02"),
  likes : ["golf", "racquetball"]
}
{
  _id : "joe",
  joined : ISODate("2012-07-02"),
  likes : ["tennis", "golf", "swimming"]
}
```

- Queremos los 5 likes más comunes

# EXPLICACIÓN \$UNWIND

```
{  
  _id : "jane",  
  joined : ISODate("2011-03-02"),  
  likes : ["golf", "racquetball"]  
}
```

- El operador \$unwind dará como salida los dos documentos siguientes:

```
{  
  _id : "jane",  
  joined : ISODate("2011-03-02"),  
  likes : "golf"  
}  
{  
  _id : "jane",  
  joined : ISODate("2011-03-02"),  
  likes : "racquetball"  
}
```

# QUEREMOS LOS 5 LIKES MÁS COMUNES

```
db.users.aggregate(  
  [  
    { $unwind : "$likes" },  
    { $group : { _id : "$likes" , number : { $sum : 1 } } },  
    { $sort : { number : -1 } },  
    { $limit : 5 }  
  ]  
)
```

# RESULTADO:

```
{
  "_id" : "golf",
  "number" : 33
},
{
  "_id" : "racquetball",
  "number" : 31
},
{
  "_id" : "swimming",
  "number" : 24
},
{
  "_id" : "handball",
  "number" : 19
},
{
  "_id" : "tennis",
  "number" : 18
}
```

Un ejemplo  
un poco más complejo

# JSON QUE USAREMOS PARA OTRO EJEMPLO

Tenemos una colección llamada zipcodes de códigos postales:

```
{
  "_id": "10280",
  "ciudad": "Madrid",
  "comunidad": "Comunidad de Madrid",
  "barrio": "Chamberí",
  "poblacion": 550074,
  "loc": [
    -74.0163,
    40.7105
  ]
}

{
  "_id": "10290",
  "ciudad": "Madrid",
  "comunidad": "Comunidad...",
  "barrio": "Fuencarral",
  "poblacion": 250074,
  "loc": [
    -73.016,
    41.710
  ]
}
```

- Queremos las comunidades con una población superior a 1 millón
- Queremos la población media de las ciudades de cada comunidad
- Queremos las ciudades más grandes y más pequeñas por comunidad

# COMUNIDADES CON UNA POBLACIÓN SUPERIOR A 1 MILLON

```
db.zipcodes.aggregate([  
  { $group: { _id: "$comunidad", totalPop:{ $sum: "$poblacion" } } },  
  { $match: { totalPop: { $gte: 1000000 } } }  
])
```

- Esta operación está formada por una fase \$group y una fase \$match.
- **\$group** agrupa los documentos por el campo “comunidad”, y calcula para cada comunidad la población total. Devuelve un documento para cada comunidad.
- El nuevo documento tiene dos campos, \_id y totalPop
- **\$match** toma estos documentos y devuelve sólo los que tienen más de 1M de totalPop
- Equivalencia en SQL:
- **SELECT comunidad, SUM(poblacion) AS totalPop FROM zipcodes GROUP BY comunidad HAVING totalPop >= (1000000)**

# POBLACIÓN MEDIA DE LAS CIUDADES DE CADA COMUNIDAD

```
db.zipcodes.aggregate([
  {$group: {_id: {comunidad: "$comunidad", ciudad: "$ciudad"}, poblacion:
{$sum: "$poblacion"}}},
  {$group: { _id: "$_id.comunidad", avgCityPop: { $avg: "$poblacion" }}}
])
```

- 2 fases group
- La primera agrupa los documentos por ciudad y comunidad y calcula la población total para cada uno
- Tras este paso los documentos son así:

```
{
  "_id" : {
    "comunidad" : "Comunidad de Madrid",
    "ciudad" : "Mostoles"
  },
  "poblacion" : 1366154
}
```



# POBLACIÓN MEDIA DE LAS CIUDADES DE CADA COMUNIDAD

- La segunda fase \$group agrupa los documentos en el pipeline por el `_id.comunidad` (campo comunidad en el documento `_id`), y usa \$avg para calcular el valor avgCityPop, sacando un documento para cada comunidad

```
{  
  "_id" : "Comunidad de Madrid",  
  "avgCityPop" : 1450335  
}
```

# CIUDADES MÁS GRANDES Y MAS PEQUEÑAS POR COMUNIDAD

```
db.zipcodes.aggregate([
  { $group: {
    _id: { comunidad: "$comunidad", ciudad: "$ciudad" },
    pop: { $sum: "$poblacion" }
  }
},
{ $sort: { pop: 1 } },
{ $group: {
  _id: "$_id.comunidad",
  biggestCityName: { $last: "$_id.ciudad" },
  biggestPop: { $last: "$pop" },
  smallestCityName: { $first: "$_id.ciudad" },
  smallestPop: { $first: "$pop" }
}
},
{ $project: {
  _id: 0,
  comunidad: "$_id",
  biggestCity: { name: "$biggestCityName", pop: "$biggestPop" },
  smallestCity: { name: "$smallestCityName", pop: "$smallestPop" }
}
}
])
```

# CON NUESTRA COLECCIÓN STUDENTS

Calcular la media de las edades de todos (para acceder a todos y no agrupar por ningún campo se usa `_id: null`)

```
db.students.aggregate([
  { $group:
    { _id: null,
      media: { $avg: "$age" }
    }
  }
])
```

# CON NUESTRA COLECCIÓN STUDENTS

Calcular la media de las edades por nacionalidad

```
db.students.aggregate([
  {
    $group:
    {
      _id: "$nationality",
      media: { $avg: "$age" }
    }
  }
])
```

# CON NUESTRA COLECCIÓN STUDENTS

Calcular la media de las notas, como las notas están en un array hay que hacer antes \$unwind

```
db.students.aggregate([
  {$unwind: "$scores"},
  {$group:
    { _id: null,
      media: {$avg: "$scores.score"}
    }
  }
])
```

```
db.students.aggregate([
  {$unwind: "$scores"},
  {$group:
    { _id: "$scores.type",
      media: {$avg: "$scores.score"}
    }
  }
])
```

|||

▼

\$unwind

☒

i

🗑

+

```
1 /**
2  * path - Path to the array field.
3  * includeArrayIndex - Optional name for index.
4  * preserveNullAndEmptyArrays - Optional
5  *   toggle to unwind null and empty values.
6  */
7 {
8   path: "$scores"
9 }
```

Output after \$unwind stage (Sample of 20 documents)

\_id: 2

name: "Corliss Zuk"

▼ scores: Object

type: "exam"

score: 67.03077096065002

age: 32

nationality: "american"

\_id: 2

name: "Corliss Zuk"

▶ scores: Object

age: 32

nationality: "american"

|||

▼

\$group

☒

i

🗑

+

```
1 /**
2  * _id - The id of the group.
3  * field1 - The first field name.
4  */
5 {
6   _id: "$scores.type",
7   media: {
8     $avg: "$scores.score"
9   }
10 }
```

Output after \$group stage (Sample of 4 documents)

\_id: "final"

media: 9.8

\_id: "homework"

media: 50.518967355882495



# Ejercicios en clase

```

{"_id":"54e23c7b28099359f566151a",
  "abv":"AL",
  "data":[
    {
      "totalPop":4040587,
      "totalHouse":1670379,
      "year":1990
    },
    {
      "totalPop":4447100,
      "totalHouse":1963711,
      "year":2000
    },
    {
      "totalPop":4779736,
      "totalHouse":2171853,
      "year":2010
    }
  ],
  "name":"Alabama",
  "region":"South",
  "areaM":52420.07,
  "center":{
    "type":"Point",
    "coordinates":[86.63333333,32.84166667]
  }
}

```

- Tenemos los datos oficiales del censo de EEUU en 3 años (1990, 2000 y 2010)
- Agrupados por estado
- Aunque en la colección hay 52 estados porque el censo de EEUU considera 2 extra (Puerto Rico y Distrito de Columbia)

- Para restaurar el dump:

```
.\mongorestore.exe -d cData path_to_census
```

- Repo:  
<https://github.com/jayrunkel/mongoDBAggWebFeb2015>



# EJERCICIOS DE CLASE

- Query que calcule el area total y el area media de los estados

```
{ "_id": "54e23c7b28099359f566151a",  
  "abv": "AL",  
  "data": [  
    {  
      "totalPop": 4040587,  
      "totalHouse": 1670379,  
      "year": 1990  
    },  
    {  
      "totalPop": 4447100,  
      "totalHouse": 1963711,  
      "year": 2000  
    },  
    {  
      "totalPop": 4779736,  
      "totalHouse": 2171853,  
      "year": 2010  
    }  
  ],  
  "name": "Alabama",  
  "region": "South",  
  "areaM": 52420.07,  
  "center": {  
    "type": "Point",  
    "coordinates": [86.63333333, 32.84166667]  
  }  
}
```

# EJERCICIOS DE CLASE

- Query que calcule el area total y el area media de los estados

```
db.cData.aggregate([
  {"$group": {"_id": null,
    "totalArea": {"$sum": "$areaM"},
    "avgArea": {"$avg": "$areaM"}}
  ])
```

# EJERCICIOS DE CLASE

- Query que calcule el area total por region, area media de los estados de esa region, número de estados en cada region y un array indicando qué estados son. Un documento por region como el siguiente:

```
{
  "_id" : "Northeast",
  "totalArea" : 181319.86,
  "avgArea" : 20146.651111111111,
  "numStates" : 9,
  "states" : [ "New Jersey", "Maine", "New
Hampshire", "Rhode Island",
"Massachusetts", "Vermont",
"Pennsylvania", "Connecticut", "New
York" ]
}
```

```
{ "_id": "54e23c7b28099359f566151a",
  "abv": "AL",
  "data": [
    {
      "totalPop": 4040587,
      "totalHouse": 1670379,
      "year": 1990
    },
    {
      "totalPop": 4447100,
      "totalHouse": 1963711,
      "year": 2000
    },
    {
      "totalPop": 4779736,
      "totalHouse": 2171853,
      "year": 2010
    }
  ],
  "name": "Alabama",
  "region": "South",
  "areaM": 52420.07,
  "center": {
    "type": "Point",
    "coordinates": [86.63333333, 32.84166667]
  }
}
```

# EJERCICIOS DE CLASE

- Query que calcule el area total por region, area media de los estados de esa region, número de estados en cada region y un array indicando qué estados son

```
db.cData.aggregate([
  {"$group" : {"_id" : "$region",
    "totalArea" : {$sum : "$areaM"},
    "avgArea" : {$avg : "$areaM"},
    "numStates" : {$sum : 1},
    "states" : {$push : "$name"}}}
])
```

# EJERCICIOS DE CLASE

- Query que calcule la población total por año ordenados de mayor a menor
- Nota: No Podemos agrupar por datos que están dentro de un array (“embedded in an array”). Tendremos que usar \$unwind

```
{ "_id": "54e23c7b28099359f566151a",  
  "abv": "AL",  
  "data": [  
    {  
      "totalPop": 4040587,  
      "totalHouse": 1670379,  
      "year": 1990  
    },  
    {  
      "totalPop": 4447100,  
      "totalHouse": 1963711,  
      "year": 2000  
    },  
    {  
      "totalPop": 4779736,  
      "totalHouse": 2171853,  
      "year": 2010  
    }  
  ],  
  "name": "Alabama",  
  "region": "South",  
  "areaM": 52420.07,  
  "center": {  
    "type": "Point",  
    "coordinates": [86.63333333, 32.84166667]  
  }  
}
```

# EJERCICIOS DE CLASE

- Query que calcule la población total por año ordenados de mayor a menor
- No Podemos agrupar por datos que están dentro de un array (“embedded in an array”). Tendremos que usar \$unwind

```
db.cData.aggregate([  
    {$unwind : "$data"},  
    {$group : {"_id" : "$data.year",  
               "totalPop" : {$sum : "$data.totalPop"}}},  
    {$sort : {"totalPop" : 1}}  
])
```

# SOLUCIÓN EN MONGODB COMPASS

localhost:27017 STANDALONE

MongoDB 4.0.4 Communi

aggregation.cData

DOCUMENTS 52 TOTAL SIZE 23.2KB AVG. SIZE 457B INDEXES 2 TOTAL SIZE 32.0KB AVG. SIZE 16.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

Enter a pipeline name...

SAVE PIPELINE



COMMENT MODE



SAMPLE MODE



AUTO PREVIEW

Unsaved changes

☰ ▾ Sunwind



Output after \$unwind stage (Sample of 20 documents)

```
1 //**
2 * path - Path to the array field.
3 * includeArrayIndex - Optional name for index.
4 * preserveNullAndEmptyArrays - Optional
5 * toggle to unwind null and empty values.
6 */
7 {
8   path: "$data"
9 }
```

```
_id: ObjectId("54e23c7b28099359f566151a")
stateNum: 1
abv: "AL"
state: "01"
data: Object
  totalPop: 4040587
  totalHouse: 1670379
  occHouse: 3948185
  year: 1990
  name: "Alabama"
```

```
_id: ObjectId("54e23c7b28099359f566151a")
stateNum: 1
abv: "AL"
state: "01"
data: Object
  totalPop: 4447100
  totalHouse: 1963711
  occHouse: 1737080
  year: 2000
```

```
_id: ObjectId("54e23c7b28099359f566151a")
stateNum: 1
abv: "AL"
state: "01"
data: Object
  totalPop: 4779736
  totalHouse: 2171853
  occHouse: 1883791
  year: 2010
```

☰ ▾ \$group



Output after \$group stage (Sample of 3 documents)

```
1 //**
2 * _id - The id of the group.
3 * field1 - The first field name.
4 */
5 {
6   "_id" : "$data.year",
7   "totalPop" : { $sum : "$data.totalPop" }
8 }
9 }
```

```
_id: 2000
totalPop: 281421906
```

```
_id: 2010
totalPop: 312471327
```

```
_id: 1990
totalPop: 248709873
```

☰ ▾ \$sort



Output after \$sort stage (Sample of 3 documents)

```
1 //**
2 * Provide any number of field/order pairs.
3 */
4 { "totalPop" : 1 }
5 }
```

```
_id: 1990
totalPop: 248709873
```

```
_id: 2000
totalPop: 281421906
```

```
_id: 2010
totalPop: 312471327
```

# EJERCICIOS DE CLASE

- Query que calcule la población totales la region del sur (south) por año ordenados de mayor a menor
- Nota: No Podemos agrupar por datos que están dentro de un array (“embedded in an array”). Tendremos que usar \$unwind
- Nota: Esta es la misma que la anterior pero solo de los estados de la region “south”

```
{ "_id": "54e23c7b28099359f566151a",  
  "abv": "AL",  
  "data": [  
    {  
      "totalPop": 4040587,  
      "totalHouse": 1670379,  
      "year": 1990  
    },  
    {  
      "totalPop": 4447100,  
      "totalHouse": 1963711,  
      "year": 2000  
    },  
    {  
      "totalPop": 4779736,  
      "totalHouse": 2171853,  
      "year": 2010  
    }  
  ],  
  "name": "Alabama",  
  "region": "South",  
  "areaM": 52420.07,  
  "center": {  
    "type": "Point",  
    "coordinates": [86.63333333, 32.84166667]  
  }  
}
```



# EJERCICIOS DE CLASE

- Query que calcule la población total de la región del sur (south) por año ordenados de mayor a menor
- No Podemos agrupar por datos que están dentro de un array (“embedded in an array”). Tendremos que usar \$unwind
- Nota: Esta es la misma que la anterior pero solo de los estados de la región “south”

```
db.cData.aggregate([  
    {$match : {"region" : "South"}},  
    {$unwind : "$data"},  
    {$group : {"_id" : "$data.year",  
                "totalPop" : {$sum : "$data.totalPop"}}},  
    {$sort : {"totalPop" : 1}}  
])
```

# EJERCICIOS DE CLASE

- Query que calcule los cambios en la población de los estados. ¿Han cambiado de población y cuánto?

```
{ "_id": "54e23c7b28099359f566151a",  
  "abv": "AL",  
  "data": [  
    {  
      "totalPop": 4040587,  
      "totalHouse": 1670379,  
      "year": 1990  
    },  
    {  
      "totalPop": 4447100,  
      "totalHouse": 1963711,  
      "year": 2000  
    },  
    {  
      "totalPop": 4779736,  
      "totalHouse": 2171853,  
      "year": 2010  
    }  
  ],  
  "name": "Alabama",  
  "region": "South",  
  "areaM": 52420.07,  
  "center": {  
    "type": "Point",  
    "coordinates": [86.63333333, 32.84166667]  
  }  
}
```

# EJERCICIOS DE CLASE

- Query que calcule los cambios en la población de los estados. ¿Han cambiado de población y cuánto?

```
db.cData.aggregate( [  
  {$unwind: "$data"},  
  {$sort: {"data.year": 1}},  
  {$group: {"_id": "$name",  
            "pop1990": {"$first": "$data.totalPop"},  
            "pop2010": {"$last": "$data.totalPop"}}},  
  {$project: {"_id": 0,  
              "name": "$_id",  
              "delta": {"$subtract": ["$pop2010", "$pop1990"]},  
              "pop1990": 1,  
              "pop2010": 1}  
  }  
])
```

# SOLUCIÓN EN MONGODB COMPASS

localhost:27017 STANDALONE

MongoDB 4.0.4 Com

aggregation.cData

DOCUMENTS 52 TOTAL SIZE 23.2KB AVG. SIZE 457B INDEXES 2 TOTAL SIZE 32.0KB AVG. SIZE 16B

Documents Aggregations Schema Explain Plan Indexes Validation

> popdeltabystate SAVE PIPELINE ... COMMENT MODE SAMPLE MODE AUTO PREVIEW

\$sort

Output after \$sort stage (Sample of 20 documents)

```
1 //**
2 * Provide any number of field/order pairs.
3 */
4 {"data.year" : 1}
5
```

```
_id: ObjectId("54e23c7b28099359f5661520")
stateNum: 34
abv: "NJ"
state: "34"
data: Object
  totalPop: 7730188
  totalHouse: 3075310
  occHouse: 7558820
  year: 1990
```

```
_id: ObjectId("54e23c7b28099359f5661525")
stateNum: 6
abv: "CA"
state: "6"
data: Object
  totalPop: 29760021
  totalHouse: 11182882
  occHouse: 29008161
  year: 1990
```

```
_id: ObjectId("54e23c7b28099359f5661528")
stateNum: 18
abv: "IN"
state: "18"
data: Object
  totalPop: 5544159
  totalHouse: 2246046
  occHouse: 5382167
  year: 1990
```

\$group

Output after \$group stage (Sample of 20 documents)

```
1 //**
2 * _id - The id of the group.
3 * field1 - The first field name.
4 */
5 {"_id" : "$name",
6  "pop1990" : {"$first" : "$data.totalPop"},
7  "pop2010" : {"$last" : "$data.totalPop"}}
8
```

```
_id: "Puerto Rico"
pop1990: 3725789
pop2010: 3725789
```

```
_id: "Tennessee"
pop1990: 4877185
pop2010: 6346105
```

```
_id: "New York"
pop1990: 17990455
pop2010: 19378102
```

\$project

Output after \$project stage (Sample of 20 documents)

```
1 //**
2 * specifications - The fields to
3 * include or exclude.
4 */
5 {"_id" : 0,
6  "name" : "$_id",
7  "delta" : {"$subtract" :
8    [{"pop2010", "pop1990"}]},
9  "pop1990" : 1,
10 "pop2010" : 1}
11
```

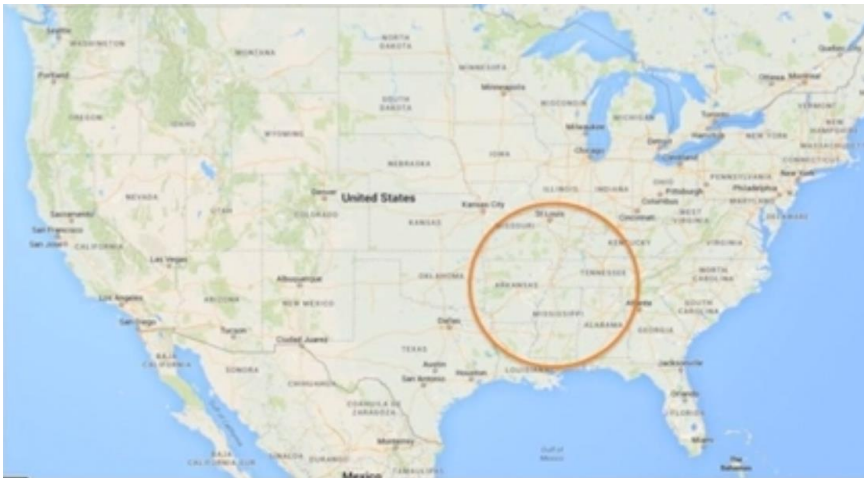
```
pop1990: 3725789
pop2010: 3725789
name: "Puerto Rico"
delta: 0
```

```
pop1990: 4877185
pop2010: 6346105
name: "Tennessee"
delta: 1468920
```

```
pop1990: 17990455
pop2010: 19378102
name: "New York"
delta: 1387647
```

# EJERCICIOS DE CLASE

- Query que compare el número de personas que viven a 500km de Memphis, TN, en 1990, 2000 y 2010
- Nota: geoNear no entra en el examen



```
{ "_id": "54e23c7b28099359f566151a",  
  "abv": "AL",  
  "data": [  
    {  
      "totalPop": 4040587,  
      "totalHouse": 1670379,  
      "year": 1990  
    },  
    {  
      "totalPop": 4447100,  
      "totalHouse": 1963711,  
      "year": 2000  
    },  
    {  
      "totalPop": 4779736,  
      "totalHouse": 2171853,  
      "year": 2010  
    }  
  ],  
  "name": "Alabama",  
  "region": "South",  
  "areaM": 52420.07,  
  "center": {  
    "type": "Point",  
    "coordinates": [86.63333333, 32.84166667]  
  }  
}
```

# EJERCICIOS DE CLASE

- Query que compare el número de personas que viven a 500km de Memphis, TN, en 1990, 2000 y 2010

```
db.cData.aggregate([
  {$geoNear : {
    "near" : {"type" : "Point", "coordinates" : [90, 35]},
    distanceField : "dist.calculated",
    maxDistance : 500000,
    includeLocs : "dist.location",
    spherical : true
  }},
  {$unwind : "$data"},
  {$group : {"_id" : "$data.year",
    "totalPop" : {"$sum" : "$data.totalPop"},
    "states" : {"$addToSet" : "$name"}}},
  {$sort : {"_id" : 1}}
])
```

# SOLUCIÓN EN MONGODB COMPASS

aggregation.cData

DOCUMENTS 52 TOTAL SIZE 23.2KB AVG. SIZE 457B INDEXES 2 TOTAL SIZE 32.0KB AVG. SIZE 16.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

> Enter a pipeline name... SAVE PIPELINE ... COMMENT MODE SAMPLE MODE AUTO PREVIEW

\$geoNear

Output after \$geoNear stage (Sample of 6 documents)

```
1 {
2   near: {
3     type: 'Point',
4     coordinates: [
5       90,
6       35
7     ]
8   },
9   distanceField: 'dist.calculated',
10  maxDistance: 50000,
11  includeLocs: 'dist.location',
12  spherical: true
13 }
```

```
_id: ObjectId("54e23c7b28099359f566152f")
stateNum: 5
name: "Arkansas"
state: "5"
region: "South"
regNum: 3
division: "West South Central"
divNum: 7
data: Array
```

```
_id: ObjectId("54e23c7b28099359f5661529")
stateNum: 28
abv: "MS"
state: "28"
data: Array
  0: Object
    totalPop: 2844658
    totalHouse: 1161953
    ncrHouse: 1846434
```

```
_id: ObjectId("54e23c7b28099359f566152a")
stateNum: 47
abv: "TN"
state: "47"
data: Array
  0: Object
    totalPop: 4877185
    totalHouse: 2026067
    ncrHouse: 4748056
```

\$unwind

Output after \$unwind stage (Sample of 18 documents)

```
1 '$data'
```

```
_id: ObjectId("54e23c7b28099359f566152f")
stateNum: 5
name: "Arkansas"
state: "5"
region: "South"
regNum: 3
division: "West South Central"
divNum: 7
data: Object
```

```
_id: ObjectId("54e23c7b28099359f566152f")
stateNum: 5
name: "Arkansas"
state: "5"
region: "South"
regNum: 3
division: "West South Central"
divNum: 7
data: Object
```

```
_id: ObjectId("54e23c7b28099359f566152f")
stateNum: 5
name: "Arkansas"
state: "5"
region: "South"
regNum: 3
division: "West South Central"
divNum: 7
data: Object
```

\$group

Output after \$group stage (Sample of 3 documents)

```
1 {
2   _id: '$data.year',
3   totalPop: {
4     $sum: '$data.totalPop'
5   },
6   states: {
7     $addToSet: '$name'
8   }
9 }
```

```
_id: 2000
totalPop: 25291421
states: Array
  0: "Kentucky"
  1: "Missouri"
  2: "Alabama"
  3: "Mississippi"
  4: "Tennessee"
  5: "Arkansas"
```

```
_id: 1990
totalPop: 22644082
states: Array
  0: "Kentucky"
  1: "Missouri"
  2: "Alabama"
  3: "Mississippi"
  4: "Tennessee"
  5: "Arkansas"
```

```
_id: 2010
totalPop: 27337350
states: Array
  0: "Kentucky"
  1: "Missouri"
  2: "Alabama"
  3: "Mississippi"
  4: "Tennessee"
  5: "Arkansas"
```

A series of vertical stripes in various shades of green and grey run down the left side of the slide. Overlaid on these stripes are several green circles of different sizes. One large circle is positioned near the top left, and several smaller circles are arranged in a descending pattern below it. The text 'MAP-REDUCE' is positioned to the right of these circles.

# MAP-REDUCE

48



# INTRODUCCIÓN A MAPREDUCE

- MapReduce es un paradigma, modelo o técnica de programación para **paralelizar** algoritmos de procesamiento de datos (los que sean paralelizables)
- Está formado por una función ***map*** seguida de una función ***reduce***
- Ambas se aplican en paralelo sobre los datos
- Ambas son funciones **puras**
  - No pueden tener efectos fuera de la función
  - No pueden hacer queries adicionales
  - No pueden usar Date, random, ...
  - Dada una entrada dan siempre la misma salida
  - <http://www.etnassoft.com/2016/06/21/las-funciones-puras-en-javascript-concepto-ejemplos-y-beneficios/>
- Se puede **distribuir** entre múltiples máquinas

# EJEMPLO DEL CONCEPTO

- Un ejemplo para entenderlo sería cómo contar las veces que ocurren todas las palabras en un libro, por ejemplo el Quijote:
- **Opción 1:** una persona cuenta las palabras y va apuntando:
  - (En, 1). (Un, 1). (Lugar, 1). (de, 1). ...
  - Y procesa todas las páginas para al final entregar un resultado
  - (En, 3498), (Un, 4543), (Lugar, 34), (de, 2343), ...
- **Opción 2:** pido a 688 personas que cuenten las palabras de cada página y cada una me entregue un resultado de su página asignada (eso sería el **Map**).
  - La primera persona me entrega (En, 4). (Un, 5). (Lugar, 1). (de, 6). ...
  - La segunda persona me entrega (Es, 2).(pues, 1). (de, 3). (saber, 2). ...
  - Etc hasta el 688
- Con el resultado lo agrego, lo reduzco, eso sería el **Reduce** obteniendo el mismo resultado que en la opción 1

# ORIGEN Y USO

- Origen by Google:
  - Jeffrey Dean and Sanjay Ghemawat: “MapReduce: Simplified Data Processing on Large Clusters,” at *6th USENIX Symposium on Operating System Design and Implementation* (OSDI), December 2004.
- Muy usado para procesamiento de datos (aunque está cayendo su uso):
  - Henry Robinson: “The Elephant Was a Trojan Horse: On the Death of Map-Reduce at Google,” *the-paper-trail.org*, June 25, 2014.
  - **Google Dumps MapReduce in Favor of New Hyper-Scale Analytics System**  
(<https://www.datacenterknowledge.com/archives/2014/06/25/google-dumps-mapreduce-favor-new-hyper-scale-analytics-system>)

# EXPLICACIÓN MAP-REDUCE

- La función **map** se encarga del mapeo o asociación y es aplicada en **paralelo** para cada ítem en la entrada de datos. Toma datos con un tipo en un dominio de datos, y devuelve una lista de pares clave-valor en un dominio diferente:

$$\text{Map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$$

- Importante  $\rightarrow$  **map()** produce una lista de pares  $(k_2, v_2)$  por cada llamada
- Después de eso, el framework de MAPREDUCE junta todos los pares con la misma clave de todas las listas y los agrupa, creando un grupo por cada una de las diferentes claves generadas (similar a como hacía la fase \$group)
- Finalmente la función **reduce** es aplicada en **paralelo** para cada grupo, produciendo una colección de valores para cada dominio

$$\text{Reduce}(k_2, \text{list}(v_2)) \rightarrow \text{list}(v_3)$$

- Por lo tanto MapReduce transforma una lista de pares (clave, valor) en una lista de valores

# MAP-REDUCE EN MONGODB

- Las operaciones map-reduce utilizan funciones JavaScript y tienen dos fases (y una extra opcional):
  - **Map**: procesa cada documento y emite uno o más objetos (key, value) para cada documento de entrada
  - **Reduce**: combina la salida del map
  - Opcionalmente puede haber una tercera fase “**finalize**” que hace unas modificaciones finales
- Al usar funciones JavaScript son más potentes que el aggregation pipeline pero en general es menos eficiente
- Funciona en particiones
- Ejemplos: <https://docs.mongodb.org/manual/tutorial/map-reduce-examples/>

# SINTAXIS

```
db.collection.mapReduce(  
    function() {emit(key,value);},  
    function (key, values) { return reduceResult; },  
    {  
        out: collection,  
        query: document,  
        sort: document,  
        limit: number  
    }  
)
```

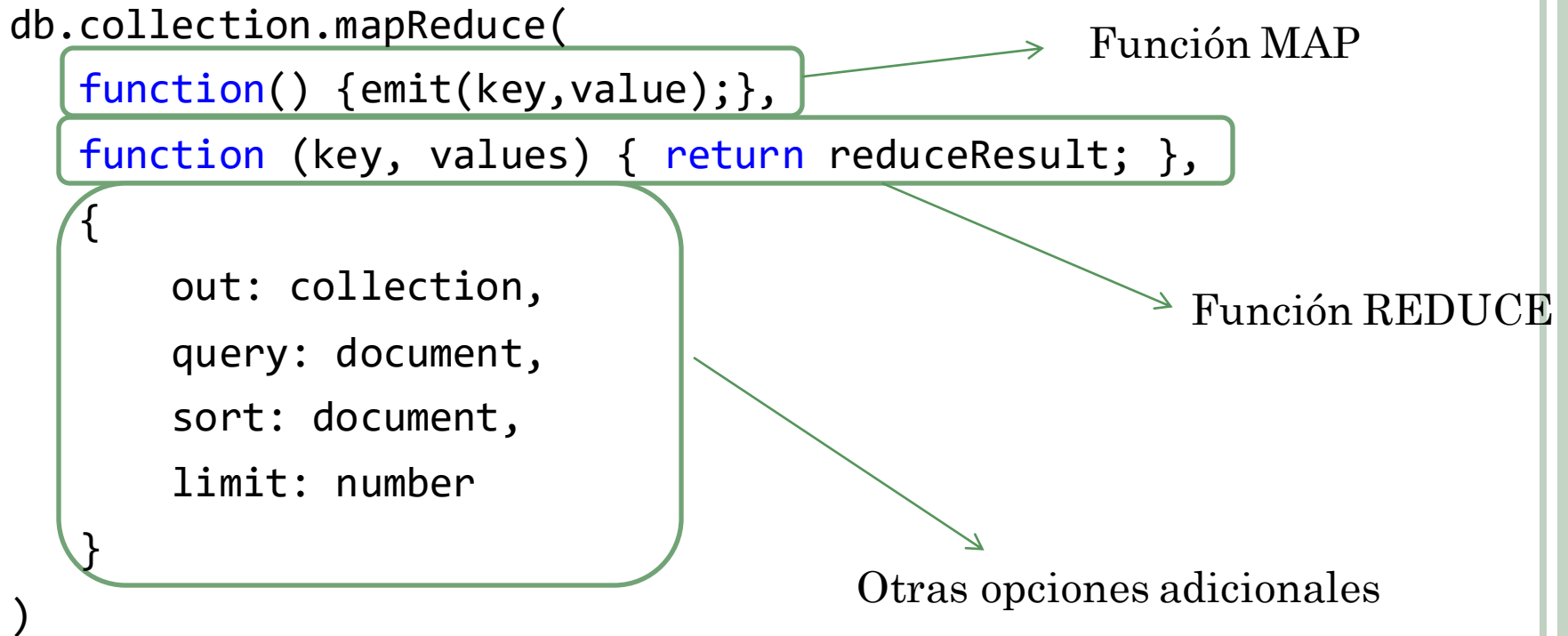
# SINTAXIS

```
db.collection.mapReduce(  
  function() {emit(key,value);},  
  function (key, values) { return reduceResult; },  
  {  
    out: collection,  
    query: document,  
    sort: document,  
    limit: number  
  }  
)
```

Función MAP

Función REDUCE

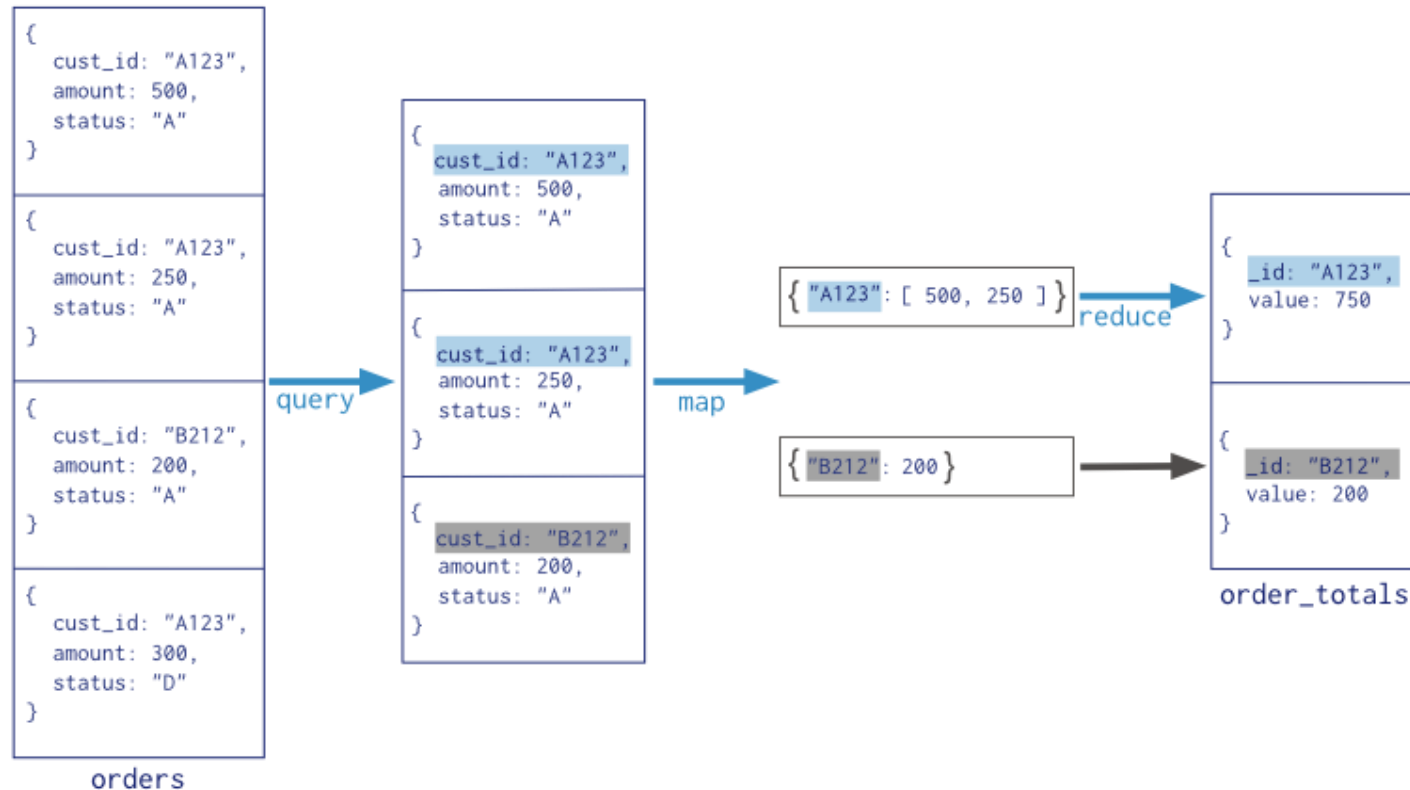
Otras opciones adicionales



# MAP REDUCE – EXPLICACIÓN VISUAL

QUEREMOS LA CANTIDAD TOTAL POR CLIENTE CON STATUS “A”

```
db.orders.mapReduce(  
  map    → function() { emit( this.cust_id, this.amount ); },  
  reduce → function(key, values) { return Array.sum( values ) },  
  {  
    query → { status: "A" },  
    output → "order_totals"  
  }  
)
```





# EXPLICACIÓN

- **Query** ha filtrado previamente los que tienen status “A”
- La función **map** pasa por todos los documentos y emite pares (key, value) con (key: cust\_id, value: amount):
  - (“A123”, 500), (“A123”, 250), (“B212”, 200)
- La función **reduce** recibe ya agregados estos pares por key y los procesa
  - Se le llama dos veces así:
    - reduce(“A123”, [500, 250])
    - reduce(“B212”, [200])
- La salida final sería:
  - { \_id: “A123”, value: 750 } y { \_id: “B212”, value: 200 }

# MISMO MAP-REDUCE

## CON LAS FUNCIONES DEFINIDAS SEPARADAS

- Nota: “**this**” se refiere al documento que se está procesando en el map en ese momento

```
var mapFunction1 = function () {  
    emit(this.cust_id, this.amount);  
};
```

```
var reduceFunction1 = function (key, values) {  
    return Array.sum(values);  
};
```

```
db.orders.mapReduce(  
    mapFunction1,  
    reduceFunction1,  
    {  
        query: { "status": "A"},  
        out: "order_totals"  
    }  
)
```

# EJEMPLO SIMPLE

Un biólogo añade una entrada a una BBDD cada vez que ve un animal en el océano.

Tenemos así una colección con N documentos:

```
{
  observationTimestamp: Date.parse("Mon, 25 Dec 1995 12:34:56 GMT"),
  family:      "Sharks",
  species:     "Carcharodon carcharias",
  numAnimals: 3
}
{
  observationTimestamp: Date.parse("Tue, 12 Dec 1995 16:17:18 GMT"),
  family:      "Sharks",
  species:     "Carcharias taurus",
  numAnimals: 4
}
```

Queremos un reporte que cuente cuantos tiburones se han visto al mes.

# EJEMPLO SIMPLE - CON SQL

```
{
  observationTimestamp: Date.parse("Mon, 25 Dec 1995 12:34:56 GMT"),
  family:      "Sharks",
  species:     "Carcharodon carcharias",
  numAnimals: 3
}
{
  observationTimestamp: Date.parse("Tue, 12 Dec 1995 16:17:18 GMT"),
  family:      "Sharks",
  species:     "Carcharias taurus",
  numAnimals: 4
}
```

```
SELECT date_trunc('month', observation_timestamp) AS observation_month,
       sum(num_animals) AS total_animals
FROM observations
WHERE family = 'Sharks'
GROUP BY observation_month;
```

# EJEMPLO SIMPLE - CON AGGREGATION FRAMEWORK

```
{
  observationTimestamp: Date.parse("Mon, 25 Dec 1995 12:34:56 GMT"),
  family:      "Sharks",
  species:     "Carcharodon carcharias",
  numAnimals: 3
}
{
  observationTimestamp: Date.parse("Tue, 12 Dec 1995 16:17:18 GMT"),
  family:      "Sharks",
  species:     "Carcharias taurus",
  numAnimals: 4
}
```

```
db.observations.aggregate([
  { $match: { family: "Sharks" } },
  { $group: {
    _id: {
      year: { $year: "$observationTimestamp" },
      month: { $month: "$observationTimestamp" }
    },
    totalAnimals: { $sum: "$numAnimals" }
  } }
]);
```

# EJEMPLO SIMPLE - CON MAPREDUCE

```
{
  observationTimestamp: Date.parse("Mon, 25 Dec 1995 12:34:56 GMT"),
  family:      "Sharks",
  species:     "Carcharodon carcharias",
  numAnimals: 3
}
{
  observationTimestamp: Date.parse("Tue, 12 Dec 1995 16:17:18 GMT"),
  family:      "Sharks",
  species:     "Carcharias taurus",
  numAnimals: 4
}
```

```
db.observations.mapReduce(
  function map() { ❷
    var year = this.observationTimestamp.getFullYear();
    var month = this.observationTimestamp.getMonth() + 1;
    emit(year + "-" + month, this.numAnimals); ❸
  },
  function reduce(key, values) { ❹
    return Array.sum(values); ❺
  },
  {
    query: { family: "Sharks" }, ❶
    out: "monthlySharkReport" ❻
  }
);
```

- The map function would be called once for each document, resulting in emit("1995-12", 3) and emit("1995-12", 4). Subsequently, the reduce function would be called with reduce("1995-12", [3, 4]), returning 7

# JSON CON EL QUE VAMOS A HACER MAS EJEMPLOS

```
{  
  _id: ObjectId("50a8240b927d5d8b5891743c"),  
  cust_id: "abc123",  
  ord_date: new Date("Oct 04, 2012"),  
  status: 'A',  
  price: 25,  
  items: [ { sku: "mmm", qty: 5, price: 2.5 },  
            { sku: "nnn", qty: 5, price: 2.5 } ]  
}
```

Stock-keeping unit o SKU (en castellano número de referencia) es un identificador usado en el comercio con el objeto de permitir el seguimiento sistémico de los productos y servicios ofrecidos a los clientes. Cada SKU se asocia con un objeto, producto, marca, servicio, cargos, etc.

# OTRO EJEMPLO

- Para las órdenes posteriores a 01/01/2012 calcular el número de órdenes y cantidad total para cada “sku”
- Calcular también cantidad media por orden por cada valor “sku”

```
{  
  _id: ObjectId("50a8240b927d5d8b5891743c"),  
  cust_id: "abc123",  
  ord_date: new Date("Oct 04, 2012"),  
  status: 'A',  
  price: 25,  
  items: [ { sku: "mmm", qty: 5, price: 2.5 },  
            { sku: "nnn", qty: 7, price: 2.5 } ]  
}
```



# MAP

- Para cada documento, esta función recorre el array “items” y asocia el sku con un nuevo valor (un objeto) que contiene “count” 1 y la cantidad (qty) de la orden
- Y **emite varios pares** (sku, valor), uno por item (fijaros que emite dentro del for)
- **Value** tiene que ser un valor de JSON, es decir puede ser un array, un objeto, un booleano, un number, un string

```
var mapFunction2 = function () {  
    for (var idx = 0; idx < this.items.length; idx++) {  
        var key = this.items[idx].sku;  
        var value = {  
            count: 1,  
            qty: this.items[idx].qty  
        };  
        emit(key, value);  
    }  
};
```

- Esta función emite pares key value. Por ejemplo para el documento de la diapositiva anterior:
- (“mmm”, {count: 1, qty: 5}) (“nnn”, {count: 1, qty:7})

# REDUCE

- Recuce recibe como siempre los pares “key, value” que emite el map, ya agregados/filtrados por key (eso lo hace Mongo)
- **Key** es el “sku” (porque la elegimos nosotros en el map)
- **Values** es **un array** cuyos elementos son los objetos {count: 1, qty: XXX} que emitió el map
- Sería como si a la función se la llamase así:
  - `reduceFunction2(“mmm”, [{count: 1, qty: 5}, {count: 1, qty: 7}, ...])`
- Esta función recude los arrays “countObjVals” a un sólo objeto “reducedVal” que contiene la suma de los campos “qty” y la suma de los campos “count”.

```
var reduceFunction2 = function (keySKU, countObjVals) {  
  var reducedVal = { count: 0, qty: 0 };  
  
  for (var idx = 0; idx < countObjVals.length; idx++) {  
    reducedVal.count += countObjVals[idx].count;  
    reducedVal.qty += countObjVals[idx].qty;  
  }  
  return reducedVal;  
};
```

# FUNCIÓN REDUCE – CONDICIONES QUE DEBE CUMPLIR

- La función debe ser pura
- La función reduce debe devolver un objeto cuyo tipo sea idéntico al tipo de valor emitido por la función map
  - Por eso muchas veces se pone un campo count: 1
  - En ese campo posteriormente se hará la agregación, si no se podría hacer `array.length`
- El orden de los elementos del array “values” (segundo argumento del reduce) no debe afectar (propiedad conmutativa)
- Debe ser una función idempotente
  - Se puede aplicar repetidas veces y no cambiaría el valor final

Ver: <https://docs.mongodb.com/manual/reference/command/mapReduce/#requirements-for-the-reduce-function>

# FINALIZE

- La función finalize recibe dos argumentos, “key, reducedVal”. Esta función modifica el reducedVal y añade un tercer campo llamado “avg” con la media y devuelve el objeto modificado.
- Se utiliza para añadir campos extra ya que como hemos visto Reduce debe devolver un objeto cuyo tipo sea idéntico al tipo de valor emitido por la función map

```
var finalizeFunction2 = function (key, reducedVal) {  
    reducedVal.avg = reducedVal.qty / reducedVal.count;  
    return reducedVal;  
};
```

# MAP-REDUCE — RESTO OPCIONES

- “Query” es para realizar el map-reduce sólo sobre la fecha que se quiere
- “Out” dice que se cree una colección “map\_reduce\_example”, y que si ya existe se haga merge (mezcla) de los contenidos

```
db.orders.mapReduce(  
    mapFunction2,  
    reduceFunction2,  
    {  
        out: { merge: "map_reduce_example" },  
        query: {  
            ord_date: { $gt: new Date('01/01/2012') }  
        },  
        finalize: finalizeFunction2  
    }  
)
```

# MAP-REDUCE INCREMENTAL I

- Si nuestra colección está creciendo y queremos hacer un map-reduce continuo, no tenemos que hacerlo con la colección completa cada vez. Se puede hacer incremental
- Será clave para el map-reduce incremental tener un timestamp (ts).
- Lo haremos en dos pasos.
- Consideremos la siguiente colección, donde length es el tiempo de sesión en segundos.
  - `db.sessions.save({ userid: "a", ts: ISODate('2011-11-03 14:17:00'), length: 95 });`
  - `db.sessions.save({ userid: "b", ts: ISODate('2011-11-03 14:23:00'), length: 110 });`
  - `db.sessions.save({ userid: "c", ts: ISODate('2011-11-03 15:02:00'), length: 120 });`
  - `db.sessions.save({ userid: "d", ts: ISODate('2011-11-03 16:45:00'), length: 45 });`
  - `db.sessions.save({ userid: "a", ts: ISODate('2011-11-04 11:05:00'), length: 105 });`
  - `db.sessions.save({ userid: "b", ts: ISODate('2011-11-04 13:14:00'), length: 120 });`
  - `db.sessions.save({ userid: "c", ts: ISODate('2011-11-04 17:00:00'), length: 130 });`
  - `db.sessions.save({ userid: "d", ts: ISODate('2011-11-04 15:37:00'), length: 65 });`
- Queremos calcular el tiempo total que ha pasado cada usuario y el tiempo medio por sesión del usuario

# MAP-REDUCE INCREMENTAL I

- Como map definimos una función que mapea el userid a un objeto que contiene el userid, total\_time, count y avg

```
var mapFunction = function () {  
    var key = this.userid;  
    var value = {  
        userid: this.userid,  
        total_time: this.length,  
        count: 1,  
        avg_time: 0  
    };  
  
    emit(key, value);  
};
```

# MAP-REDUCE INCREMENTAL I

- Como reduce definimos una función que recibe key y values y calcula el tiempo total y la cuenta total

```
var reduceFunction = function (key, values) {  
  
    var reducedObject = {  
        userid: key,  
        total_time: 0,  
        count: 0,  
        avg_time: 0  
    };  
  
    values.forEach(function (value) {  
        reducedObject.total_time += value.total_time;  
        reducedObject.count += value.count;  
    })  
    );  
    return reducedObject;  
};
```



# MAP-REDUCE INCREMENTAL I

- Como finalize definimos una función que recibe key y reducedvalue y calcula además el average

```
var finalizeFunction = function (key, reducedValue) {  
  
    if (reducedValue.count > 0)  
        reducedValue.avg_time = reducedValue.total_time / reducedValue.count;  
  
    return reducedValue;  
};
```

# MAP-REDUCE INCREMENTAL I

- Hacemos el map-reduce con las tres funciones y la salida la mandamos a una colección `session_stat`. Si la colección existe la reemplaza

```
db.sessions.mapReduce(mapFunction,  
                      reduceFunction,  
                      {  
                        out: "session_stat",  
                        finalize: finalizeFunction  
                      }  
                      )
```

# MAP-REDUCE INCREMENTAL I

- Ahora la colección va creciendo. Supongamos que metemos estos datos:

```
db.sessions.save({ userid: "a", ts: ISODate('2011-11-05 14:17:00'), length: 100 });
db.sessions.save({ userid: "b", ts: ISODate('2011-11-05 14:23:00'), length: 115 });
db.sessions.save({ userid: "c", ts: ISODate('2011-11-05 15:02:00'), length: 125 });
db.sessions.save({ userid: "d", ts: ISODate('2011-11-05 16:45:00'), length: 55 });
```

- Podremos hacer el siguiente map-reduce incremental así. Fijarse en el operador “reduce” que se le pasa a “out”. Hará que se reduzca no sólo los nuevos datos sino también los ya existentes en la colección session\_stat

```
db.sessions.mapReduce(mapFunction,
                      reduceFunction,
                      {
                        query: { ts: { $gt: ISODate('2011-11-05 00:00:00') } },
                        out: { reduce: "session_stat" },
                        finalize: finalizeFunction
                      }
                      );
```

# COMO HACER LA FUNCIÓN MAP PASO A PASO Y DEPURANDO TODO

- Para map ver <https://docs.mongodb.com/manual/tutorial/troubleshoot-map-function/>
- Para Reduce ver
- <https://docs.mongodb.com/manual/tutorial/troubleshoot-reduce-function/>
- También se pueden ir poniendo logs con la función print() y printjson() y esos logs salen por la consola del mongod (es decir los imprime la base de datos)

# HERRAMIENTA ONLINE PARA PROBAR

- <http://targetprocess.github.io/mongo-mapreduce-debug-online/>
- En esa web se puede probar paso a paso
- El map, reduce y finalize se pasan como valores de un objeto
- Poniendo en la función map o reduce la palabra “debugger;” si tenemos las dev-tools abiertas (F12) se para y podemos ejecutar paso a paso



# UN MAPREDUCE EN NUESTRA COLECCIÓN STUDENTS

# ENUNCIADO

- Recordemos como era la colección

```
{
  "_id" : 0,
  "name" : "aimee Zank Peter",
  "scores" : [
    {
      "type" : "quiz",
      "score" : 11.78273309957772
    },
    {
      "type" : "homework",
      "score" : 6.676176060654615
    }, {
      "type" : "final",
      "score" : 9.8
    },
    {
      "type" : "final",
      "score" : 9.8
    }
  ],
  "age" : 30,
  "nationality" : "american"
}
```

# ENUNCIADO

- Calcular la nota media de los distintos tipos de “scores” que hay
- Es decir la media de los homework, exam, final, ...
- Consejos:
- Cuidado si hemos añadido estudiantes sin el campo “scores”



# CON NUESTRA COLECCIÓN STUDENTS - MAP

```
var mapFunction2 = function () {  
  if (!this.scores) {  
    print("ERROR con el doc: " + this._id);  
    return;  
  } else {  
    for (var idx = 0; idx < this.scores.length; idx++) {  
      var key = this.scores[idx].type;  
      var value = this.scores[idx].score;  
      emit(key, value);  
    }  
  }  
};
```

# CON NUESTRA COLECCIÓN STUDENTS - REDUCE

```
var reduceFunction2 = function (key, value) {  
  var reducedVal = 0;  
  var bad_values = 0;  
  for (var idx = 0; idx < value.length; idx++) {  
    if (typeof value[idx] === "number") {  
      reducedVal += value[idx];  
    } else {  
      bad_values +=1;  
      print("ERROR con el value de la key " + key);  
      print("El value es")  
      printjson(value[idx]);  
    }  
  }  
  reducedVal = reducedVal / (value.length-bad_values);  
  return reducedVal;  
};
```

# CON NUESTRA COLECCIÓN STUDENTS - MAPREDUCE

```
db.students.mapReduce(  
    mapFunction2,  
    reduceFunction2,  
    {  
        out: { inline: 1 }  
    }  
)
```



# SINGLE PURPOSE AGGREGATION OPERATIONS

# SINGLE PURPOSE AGGREGATION OPERATIONS

- Mongo también provee los comandos `db.collection.count()`, `db.collection.group()`, `db.collection.distinct()`
- Todos estas operaciones agregan documentos de una colección, pero son menos flexibles que las otras opciones de agregación
- No funciona en particiones
- **count(query, options):**
  - Devuelve el número de documentos que encajan con la query.
  - Ejemplo:
  - `db.orders.count( { ord_dt: { $gt: new Date('01/01/2012') } } )`
- **distinct(field, query):**
  - Encuentra los valores diferentes de un campo específico en una colección y devuelve los resultados en un array
  - `db.inventory.distinct( "dept" )`

- **group({ key, reduce, initial [, keyf] [, cond] [, finalize] }):**
  - Agrupa documentos en una colección por la “key” y hace agregaciones simples. Equivalente a SELECT <...> GROUP BY de SQL.
  - \$group del aggregation pipeline hace lo mismo pero es más potente.
- Ejemplo:

```
db.orders.group(  
  {  
    key: { ord_dt: 1, 'item.sku': 1 },  
    cond: { ord_dt: { $gt: new Date('01/01/2012') } },  
    reduce: function (curr, result) {  
      result.total += curr.item.qty;  
    },  
    initial: { total: 0 }  
  }  
)
```



# AUTOEVALUACIÓN

87

- Given the following collection:

> db.stuff.find()

```
{ "_id" : ObjectId("50b26f9d80a78af03b5163c8"), "a" : 1, "b" : 1, "c" : 1 }
{ "_id" : ObjectId("50b26fb480a78af03b5163c9"), "a" : 2, "b" : 2, "c" : 1 }
{ "_id" : ObjectId("50b26fbf80a78af03b5163ca"), "a" : 3, "b" : 3, "c" : 1 }
{ "_id" : ObjectId("50b26fcd80a78af03b5163cb"), "a" : 3, "b" : 3, "c" : 2 }
{ "_id" : ObjectId("50b26fd380a78af03b5163cc"), "a" : 3, "b" : 5, "c" : 3 }
{ "_id" : ObjectId("50b27f7080a78af03b5163cd"), "a" : 3, "b" : 3, "c" : 2 }
```

- And the following aggregation query:

```
db.stuff.aggregate([{$group:
  { _id:
    { 'moe' : '$a',
      'larry' : '$b',
      'curly' : '$c'
    }
  }
}])
```

- How many documents will be in the result set?

- 2
- 3
- 4
- 5
- 6



- Given the following collection:

> db.stuff.find()

```
{ "_id" : ObjectId("50b26f9d80a78af03b5163c8"), "a" : 1, "b" : 1, "c" : 1 }  
{ "_id" : ObjectId("50b26fb480a78af03b5163c9"), "a" : 2, "b" : 2, "c" : 1 }  
{ "_id" : ObjectId("50b26fbf80a78af03b5163ca"), "a" : 3, "b" : 3, "c" : 1 }  
{ "_id" : ObjectId("50b26fcd80a78af03b5163cb"), "a" : 3, "b" : 3, "c" : 2 }  
{ "_id" : ObjectId("50b26fd380a78af03b5163cc"), "a" : 3, "b" : 5, "c" : 3 }  
{ "_id" : ObjectId("50b27f7080a78af03b5163cd"), "a" : 3, "b" : 3, "c" : 2 }
```

- And the following aggregation query:

```
db.stuff.aggregate([{$group:  
  {_id:  
    {'moe': '$a',  
      'larry': '$b',  
      'curly': '$c'  
    }  
  }  
}])
```

- How many documents will be in the result set?
- 2
- 3
- 4
- 5 // toma todos los documentos que tienen la combinación a,b,c diferentes
- 6

- Given the following collection:

> db.stuff.find()

```
{ "_id" : ObjectId("50b26f9d80a78af03b5163c8"), "a" : 1, "b" : 1, "c" : 1 }  
{ "_id" : ObjectId("50b26fb480a78af03b5163c9"), "a" : 2, "b" : 2, "c" : 1 }  
{ "_id" : ObjectId("50b26fbf80a78af03b5163ca"), "a" : 3, "b" : 3, "c" : 1 }  
{ "_id" : ObjectId("50b26fcd80a78af03b5163cb"), "a" : 3, "b" : 3, "c" : 2 }  
{ "_id" : ObjectId("50b26fd380a78af03b5163cc"), "a" : 3, "b" : 5, "c" : 3 }  
{ "_id" : ObjectId("50b27f7080a78af03b5163cd"), "a" : 3, "b" : 3, "c" : 2 }
```

- And the following aggregation query:

```
db.stuff.aggregate([{$group:{_id:'$c'}}])
```

- How many documents will be in the result set?
  - 2
  - 3
  - 4
  - 5
  - 6

- Given the following collection:

> db.stuff.find()

```
{ "_id" : ObjectId("50b26f9d80a78af03b5163c8"), "a" : 1, "b" : 1, "c" : 1 }  
{ "_id" : ObjectId("50b26fb480a78af03b5163c9"), "a" : 2, "b" : 2, "c" : 1 }  
{ "_id" : ObjectId("50b26fbf80a78af03b5163ca"), "a" : 3, "b" : 3, "c" : 1 }  
{ "_id" : ObjectId("50b26fcd80a78af03b5163cb"), "a" : 3, "b" : 3, "c" : 2 }  
{ "_id" : ObjectId("50b26fd380a78af03b5163cc"), "a" : 3, "b" : 5, "c" : 3 }  
{ "_id" : ObjectId("50b27f7080a78af03b5163cd"), "a" : 3, "b" : 3, "c" : 2 }
```

- And the following aggregation query:

```
db.stuff.aggregate([{$group:{_id:'$c'}}])
```

- How many documents will be in the result set?

- 2
- 3
- 4
- 5
- 6

- **What does the following aggregate query perform?**

```
db.posts.aggregate( [  
  { $match : { likes : { $gt : 100, $lte : 200 } } },  
  { $group: { _id: null, count: { $sum: 1 } } }  
]);
```

- **A** - Calculates the number of posts with likes between 100 and 200
- **B** - Groups the posts by number of likes (101, 102, 103) by adding 1 every time
- **C** - Fetches the posts with likes between 100 and 200 and sets their `_id` as null
- **D** - Fetches the posts with likes between 100 and 200, sets the `_id` of the first document as null and then increments it 1 every time

- **What does the following aggregate query perform?**

```
db.posts.aggregate( [ { $match : { likes : { $gt : 100, $lte : 200 } } }, { $group: { _id: null, count: { $sum: 1 } } } ] );
```

- **A - Calculates the number of posts with likes between 100 and 200**
- **B - Groups the posts by number of likes (101, 102, 103) by adding 1 every time**
- **C - Fetches the posts with likes between 100 and 200 and sets their \_id as null**
- **D - Fetches the posts with likes between 100 and 200, sets the \_id of the first document as null and then increments it 1 every time**
- **Explanation**
- The above query basically matches all the documents having likes between 100 and 200. After that, it just specifies that aggregation is not to be done with any specific column (\_id:null) and increments the count every time. Thus calculating the total such posts.

- **Consider that you have a collection called `population` which has fields `state` and `city`. Which of the following query will calculate the total population by city?**
- **A** - `db.population.aggregate( [{ $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } } ] )`
- **B** - `db.population.aggregate( [{ $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: 1 } } } ] )`
- **C** - `db.population.aggregate( [{ $group: { _id: { state: "$state", city: "$city" }, pop: { "$pop": 1 } } } ] )`
- **D** - `db.population.aggregate( [{ $group: { _id: { city: "$city" }, pop: { $sum: "$pop" } } } ] )`

- Consider that you have a collection called **population** which has fields **state** and **city**. Which of the following query will calculate the total population by city?
- **A** - `db.population.aggregate( [{ $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } } ] )`
- **B** - `db.population.aggregate( [{ $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: 1 } } } ] )`
- **C** - `db.population.aggregate( [{ $group: { _id: { state: "$state", city: "$city" }, pop: { "$pop": 1 } } } ] )`
- **D** - `db.population.aggregate( [{ $group: { _id: { city: "$city" }, pop: { $sum: "$pop" } } } ] )`
- Explanation
- You have to give state and city as the key to group by and then calculate the sum of the population in each city.

Given the following collection:

```
> db.fun.find()
```

```
{ "_id" : 0, "a" : 0, "b" : 0, "c" : 21 }  
{ "_id" : 1, "a" : 0, "b" : 0, "c" : 54 }  
{ "_id" : 2, "a" : 0, "b" : 1, "c" : 52 }  
{ "_id" : 3, "a" : 0, "b" : 1, "c" : 17 }  
{ "_id" : 4, "a" : 1, "b" : 0, "c" : 22 }  
{ "_id" : 5, "a" : 1, "b" : 0, "c" : 5 }  
{ "_id" : 6, "a" : 1, "b" : 1, "c" : 87 }  
{ "_id" : 7, "a" : 1, "b" : 1, "c" : 97 }
```

○ And the following aggregation query

```
db.fun.aggregate([  
  {$group:{_id:{a:"$a", b:"$b"}, c:{$max:"$c"}}},  
  {$group:{_id:"$_id.a", c:{$min:"$c"}}}  
])
```

What values are returned?

- 17 and 54
- 97 and 21
- 54 and 5
- 52 and 22



Given the following collection:

```
> db.fun.find()
```

```
{ "_id" : 0, "a" : 0, "b" : 0, "c" : 21 }  
{ "_id" : 1, "a" : 0, "b" : 0, "c" : 54 }  
{ "_id" : 2, "a" : 0, "b" : 1, "c" : 52 }  
{ "_id" : 3, "a" : 0, "b" : 1, "c" : 17 }  
{ "_id" : 4, "a" : 1, "b" : 0, "c" : 22 }  
{ "_id" : 5, "a" : 1, "b" : 0, "c" : 5 }  
{ "_id" : 6, "a" : 1, "b" : 1, "c" : 87 }  
{ "_id" : 7, "a" : 1, "b" : 1, "c" : 97 }
```

- And the following aggregation query

```
db.fun.aggregate([  
  {$group:{_id:{a:"$a", b:"$b"}, c:{$max:"$c"}}},  
  {$group:{_id:"$_id.a", c:{$min:"$c"}}}  
])
```

What values are returned?

- 17 and 54
- 97 and 21
- 54 and 5
- 52 and 22

- Dados los datos de población por código postal siguientes:

```
{  
  "city" : "FISHERS ISLAND",  
  "loc" : [  
    -72.017834,  
    41.263934  
  ],  
  "pop" : 329,  
  "state" : "NY",  
  "_id" : "06390"  
}
```

- Escribe una expresión de agregación que calcule la población media por código postal del estado. El código postal es el campo `_id` y es único. La colección se llama `zips` y la clave de resultado que se pide `average_pop`

- Dados los datos de población por código postal siguientes:

```
{  
  "city" : "FISHERS ISLAND", "loc" : [ -72.017834, 41.263934 ],  
  "pop" : 329, "state" : "NY", "_id" : "06390"  
}
```

- Escribe una expresión de agregación que calcule la población media por estado. El código postal es el campo `_id` y es único. La colección se llama `zip` y la clave de resultado que se pide `average_pop`
- 

```
db.zip.aggregate([  
  { $group:  
    {  
      _id:"$state",  
      average_pop: { $avg: "$pop" }  
    }  
  }  
])
```