Operaciones de agregación

Varias operaciones de agregación

- Operaciones de agregación de un solo propósito
- Agregation pipeline: El marco de trabajo de agregación
- Operaciones de map-reduce

Operaciones de agregación de un solo propósito

Reference > mongo Shell Methods > Collection Methods > db.collection.count()

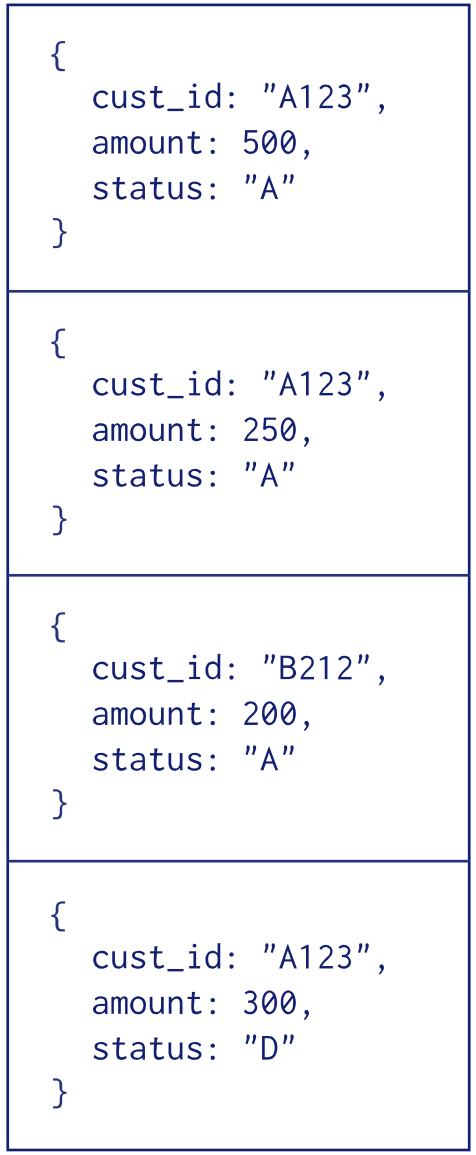
db.collection.count()

Reference > mongo Shell Methods > Collection Methods > db.collection.distinct()

db.collection.distinct() ¶

Collection

```
db.orders.distinct( "cust_id" )
```



db.collection.distinct()

distinct → ["A123", "B212"]

```
> db.movieDetails.findOne()
{
        "_id" : ObjectId("569190ca24de1e0ce2dfcd4f"),
        "title" : "Once Upon a Time in the West",
        "year" : 1968,
        "rated" : "PG-13",
        "released" : ISODate("1968-12-21T05:00:00Z"),
        "runtime" : 175,
        "countries" : [
                "Italy",
                "USA",
                "Spain"
        ],
        "genres" : [
                "Western"
        ],
        "director" : "Sergio Leone",
        "writers" : [
                "Sergio Donati",
                "Sergio Leone",
                "Dario Argento",
                "Bernardo Bertolucci",
                "Sergio Leone"
```

db.movieDetails.distinct("title", {"runtime":{$gt:180}})

```
> db.movieDetails.distinct("title", {"runtime":{$gt:180}})
[
        "Once Upon a Time in America",
        "Lagaan: Once Upon a Time in India",
        "AC/DC: Plug Me In",
        "Swallowed Land DX",
        "Ek Aur Ek Gyarah: By Hook or by Crook",
        "Muqaddar Ka Sikandar",
        "Roop Ki Rani Choron Ka Raja",
        "Heremias: Unang aklat - Ang alamat ng prinsesang bayawak",
        "The Godfather: Part II",
        "Attack on Terror: The FBI vs. the Ku Klux Klan",
        "Dilwale Dulhania Le Jayenge",
        "Tie Xi Qu: West of the Tracks",
        "TS Playground 5",
        "The Question of God: Sigmund Freud & C.S. Lewis",
        "El Cid",
        "Main Prem Ki Diwani Hoon",
        "The Decline of the Century: Testament L.Z."
]
```

El marco de trabajo de agregación

Introducción

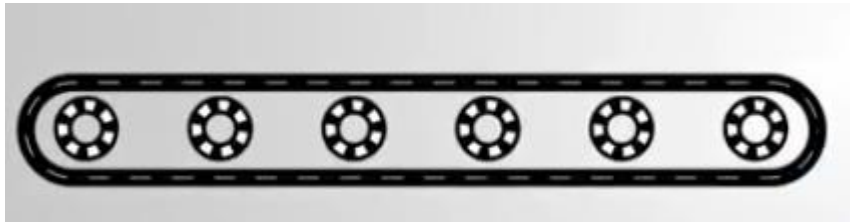
El marco de trabajo de agregación es el lenguaje avanzado de consulta de MongoDB.

Permite combinar y transformar datos a partir de varios documentos para generar nueva información que no está disponible en un solo documento.

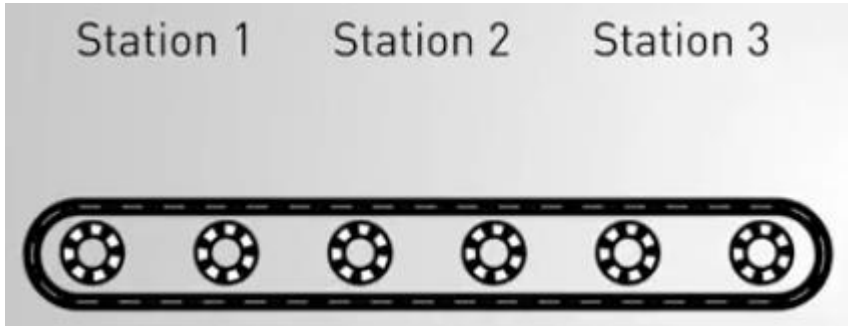
Ejemplos:

- Determinar ventas mensuales
- Determinar ventas por productos
- Total de órdenes por cliente

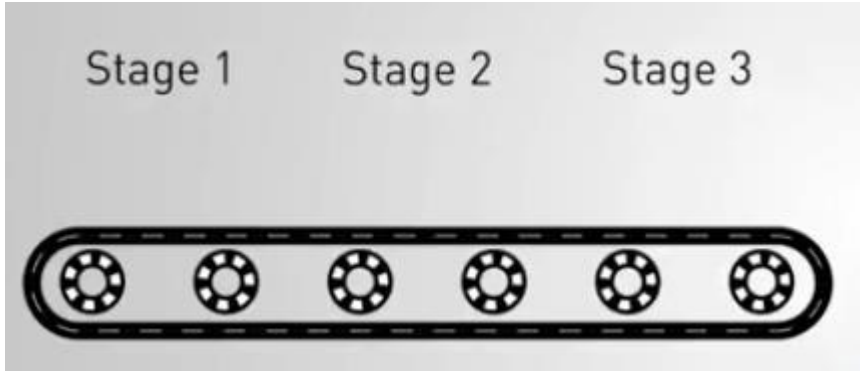
Pipelines



Piense en un pipeline como una línea de ensamblaje transportadora en una fábrica

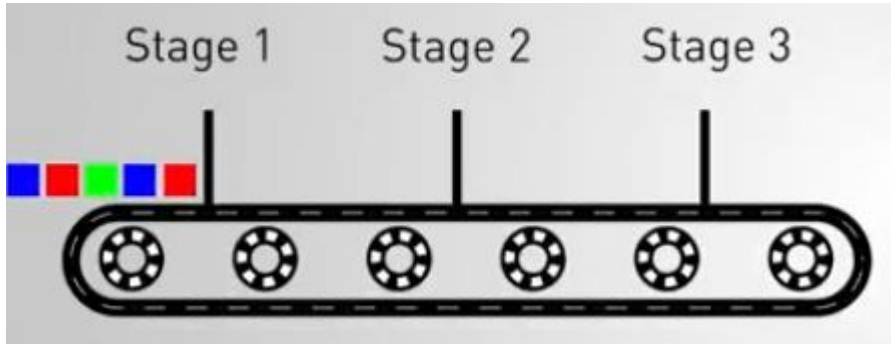


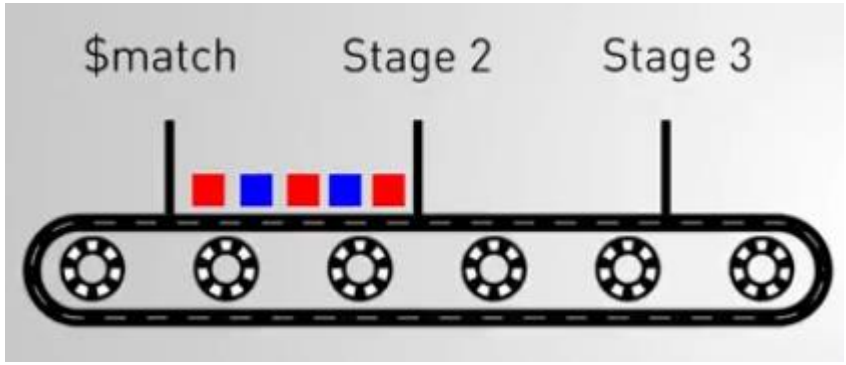
A lo largo de la línea hay un conjunto de estaciones de ensamblaje.



Esas estaciones son las etapas (stages)

Un pipeline

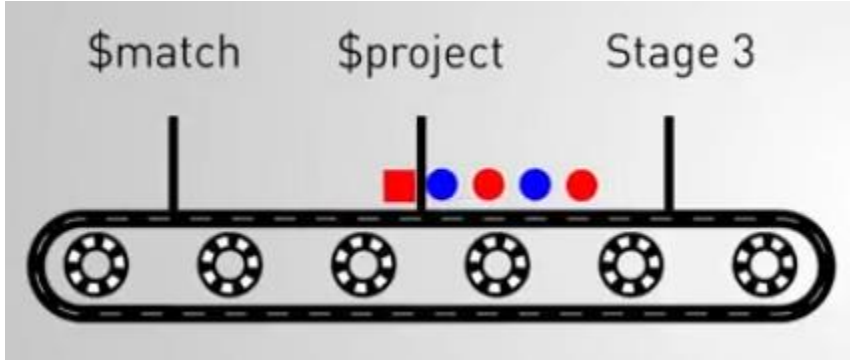




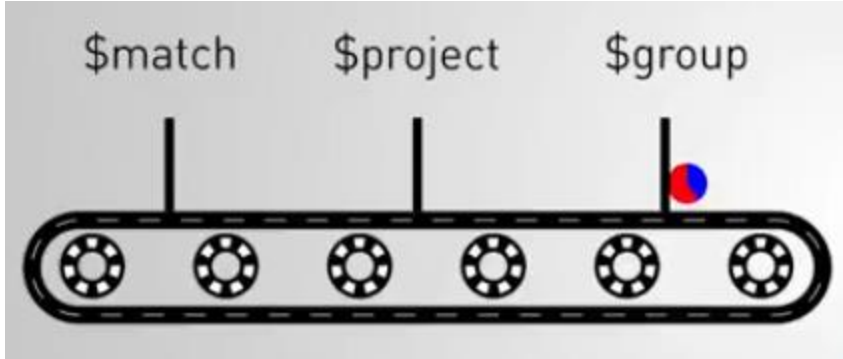
Los documentos fluyen a través de la línea transportadora

La primera etapa a la que llamamos $match permite que solamente sigan en la línea algunos tipos de documentos

Una segunda y tercera etapa



La segunda etapa, a la que llamamos $project, produce una transformación de los documentos. Podemos transformar nuestros datos en la forma que deseamos



La tercera etapa $group (una de las más poderosas) produce un único documento de salida a partir de los Documentos de entrada

Pipelines

Al llamar al marco de trabajo de agregación definimos un pipeline

Un pipeline es una composición de etapas

- Los documentos fluyen a través de las etapas
- Las etapas son configurables para producir las transformaciones deseadas. Cada paso o etapa ejecuta una sola operación sobre los documentos de entrada para transformarlos y generar nuevo(s) documento(s)
- Con pocas excepciones, las etapas pueden configurarse de muchas maneras y tantas como se requieran para lograr la transformación esperada de los datos

Estructura y sintaxis de una agregación

```
db.userColl.aggregate([{ stage1 }, { stage2 }, { …stageN }], {options})
```

Una sintaxis simple, confiable y repetitiva

Cada etapa es un objeto JSON de pares llave valor

Opciones que pueden ser pasadas al pipeline (especificar si puede hacerse uso de disco, ver el explain plan de la agregación para ver si está usando índices, si el servidor optimizó el pipeline)

Un ejemplo sencillo: Un documento de la colección solarSystem

```
{
        "_id" : ObjectId("59a06674c8df9f3cd2ee7d54
        "name" : "Earth", "type" : "Terrestrial planet",
        "orderFromSun" : 3,
        "radius" : {"value" : 6378.137, "units" : "km"},
        "mass" : {"value" : 5.9723e+24, "units" : "kg"},
        "sma" : {"value" : 149600000, "units" : "km"  },
        "orbitalPeriod" : {"value" : 1,"units" : "years"},
        "eccentricity" : 0.0167,
        "meanOrbitalVelocity" : {"value" : 29.78,"units" : "km/sec"},
        "rotationPeriod" : {"value" : 1,"units" : "days"},
        "inclinationOfAxis" : {"value" : 23.45,"units" : "degrees" },
        "meanTemperature" : 15,
        "gravity" : {"value" : 9.8, "units" : "m/s^2"},
        "escapeVelocity" : {"value" : 11.18,"units" : "km/sec" },
        "meanDensity" : 5.52,"atmosphericComposition" : "N2+O2",
        "numberOfMoons" : 1, "hasRings" : false,
        "hasMagneticField" : true
}
```

Un ejemplo sencillo

```
1   // simple first example
2   db.solarSystem.aggregate([{
3     "$match": {
4       "atmosphericComposition": { "$in": [/O2/] },
5       "meanTemperature": { $gte: -40, "$lte": 40 }
6     }
7   }, {
8     "$project": {
9       "_id": 0,
10      "name": 1,
11      "hasMoons": { "$gt": ["$numberOfMoons", 0] }
12    }
13  }], { "allowDiskUse": true});
14
```

{ "name" : "Earth", "hasMoons" : true }

Las etapas y las opciones

```
"$match": {
  "atmosphericComposition": { "$in": [/O2/] },
  "meanTemperature": { $gte: -40, "$lte": 40 }
}
```

La composición atmosférica contiene oxígeno y la temperatura promedio está en ese rango

```
 {
"$project": {
  "_id": 0,
  "name": 1,
  "hasMoons": { "$gt": ["$numberOfMoons", 0] }
}
```

Modifica el documento y calcula un nuevo valor

```
{ "allowDiskUse": true}
```

El objeto para indicar las opciones

Composición de una etapa

Cada etapa está compuesta por operadores o expresiones

Los operadores pueden referirse a operadores de consulta o etapas de una agregación.

- En el ejemplo, $match y $project son operadores de agregación y $in, $gte, $lte, son operadores de consulta.
- Los operadores siempre aparecen en posiciones claves de un documento.

Las expresiones son como funciones: brindamos argumentos y producen resultados. Las expresiones pueden tomar un solo argumento o un arreglo de argumentos.

- Las expresiones pueden combinarse para brindar transformaciones de datos
- En la etapa $project, $gt es una expresión y sus argumentos se suministran en un arreglo
- $numberOfMoons entre comillas también es una expresión.
- Las expresiones siempre aparecen en la posición del valor en el par llave-valor

Operadores de agregación

Operadores de agregación

```
1   // simple first example
2   db.solarSystem.aggregate([{
3     "$match": {
4       "atmosphericComposition": { "$in": [/O2/] },
5       "meanTemperature": { $gte: -40, "$lte": 40 }
6     }
7   }, {
8     "$project": {
9       "_id": 0,
10      "name": 1,
11      "hasMoons": { "$gt": ["$numberOfMoons", 0] }
12    }
13  }], { "allowDiskUse": true});
14
```

Query operators

expresiones

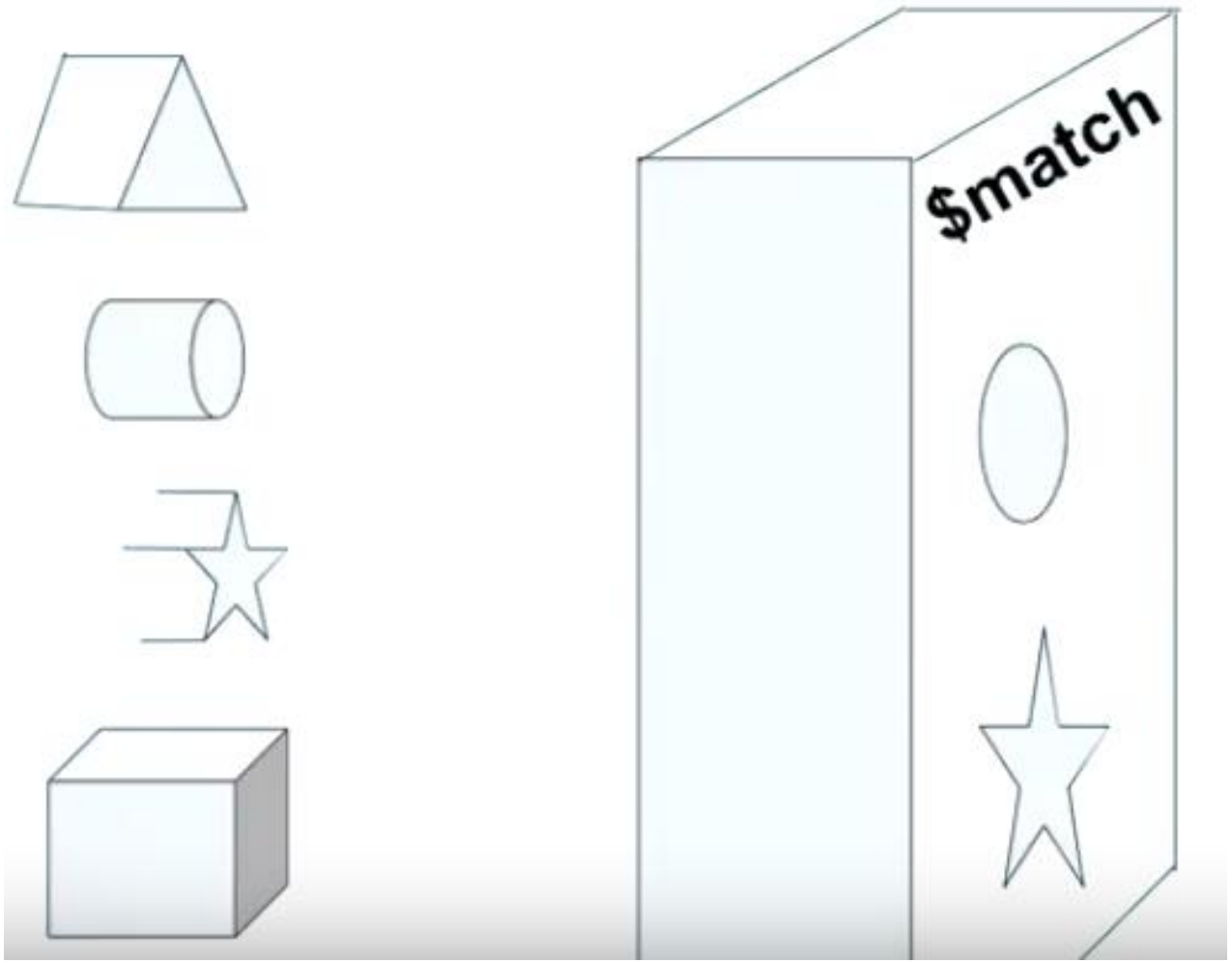
Filtrar documentos con $match

- La mayoría de las veces, esta etapa aparece como primera etapa del pipeline.
- Pueden aparecer varios $match en el pipeline.
- La sintaxis básica de $match

```
db.solarSystem.aggregate([{
  $match: { }
}])
```

- Después de $match aparecen otros operadores de agregación. Algunos de ellos no.
- $match utiliza los operadores de consulta de MongoDB.

$match



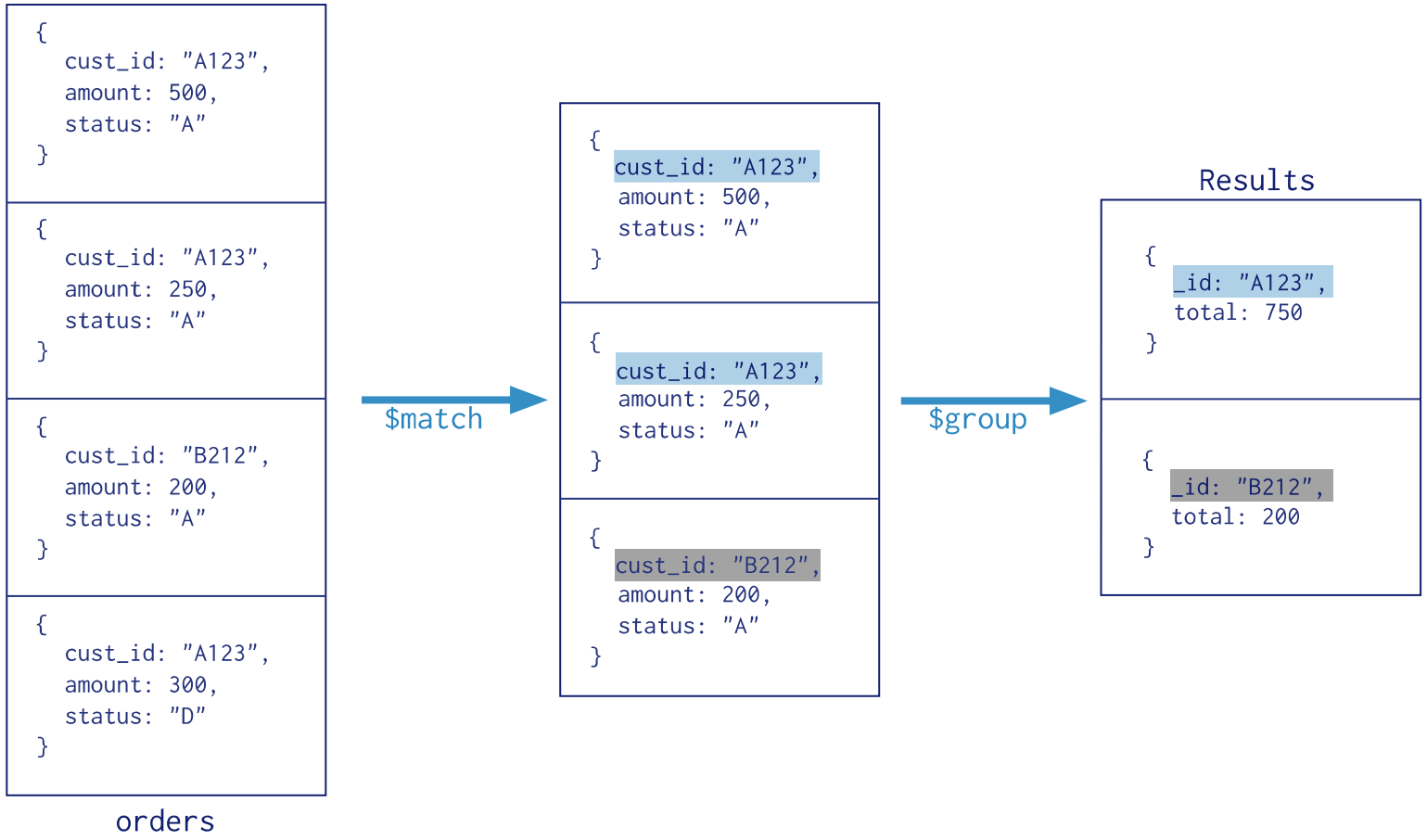
Ejemplos

```
// $match all celestial bodies, not equal to Star
db.solarSystem.aggregate([{
  "$match": { "type": { "$ne": "Star" } }
}]).pretty()

// same query using find command
db.solarSystem.find({ "type": { "$ne": "Star" } }).pretty();

// count the number of matching documents
db.solarSystem.count();

// using $count
db.solarSystem.aggregate([{
  "$match": { "type": { "$ne": "Star"} }
}, {
  "$count": "planets"
}]);

// matching on value, and removing ``_id`` from projected document
db.solarSystem.find({"name": "Earth"}, {"_id": 0});
```

Otro ejemplo

Collection

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



Puntualizando cosas

Algunas etapas del pipeline toman una expresión de pipeline como operando.

- Las expresiones del pipeline especifican la transformación que va a aplicarse a los documentos de entrada
- Las expresiones cumplen con la estructura de un documento `({key:value})` y pueden contener otra expresión

Las expresiones del pipeline pueden operar solamente con el documento corriente en el pipeline y no pueden referirse a datos de otros documentos: brindan la transformación de documentos **in-memory**

Generalmente las expresiones son **stateless** y son evaluadas en el momento que son vistas por el proceso de agregación, con la excepción de las expresiones "accumulators"

Los acumuladores (totals, maximums, minimums, etc.), usados en la etapa de `$group`, mantienen su estado a medida que los documentos pasan por el pipeline

Las etapas (stages) del marco de trabajo de agregación

db.collection.aggregate([{ <stage> }, ...])

$addFields	Adds new fields to documents. Similar to $project, $addFields reshapes each document in the stream; specifically, by adding new fields to to output documents that contain both the existing fields from the input documents and the newly added fields.

```
{
  _id: 1,
  student: "Maya",
  homework: [ 10, 5, 10 ],
  quiz: [ 10, 8 ],
  extraCredit: 0
}
{
  _id: 2,
  student: "Ryan",
  homework: [ 5, 6, 5 ],
  quiz: [ 8, 8 ],
  extraCredit: 8
}
```

```
db.scores.aggregate( [
   {
     $addFields: {
       totalHomework: { $sum: "$homework" } ,
       totalQuiz: { $sum: "$quiz" }
     }
   },
   {
     $addFields: { totalScore:
       { $add: [ "$totalHomework", "$totalQuiz", "$extraCredit" ] } }
   }
] )
```

```
{
  "_id" : 1,
  "student" : "Maya",
  "homework" : [ 10, 5, 10 ],
  "quiz" : [ 10, 8 ],
  "extraCredit" : 0,
  "totalHomework" : 25,
  "totalQuiz" : 18,
  "totalScore" : 43
}
{
  "_id" : 2,
  "student" : "Ryan",
  "homework" : [ 5, 6, 5 ],
  "quiz" : [ 8, 8 ],
  "extraCredit" : 8,
  "totalHomework" : 16,
  "totalQuiz" : 16,
  "totalScore" : 40
}
```

$bucket

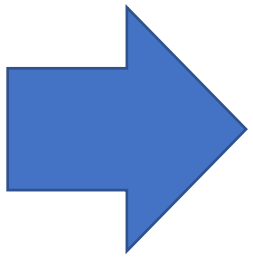
Categorizes incoming documents into groups, called buckets, based on a specified expression and bucket boundaries.

```
{ "_id" : 1, "title" : "The Pillars of Society", "artist" : "Grosz", "year" : 1926,
    "price" : NumberDecimal("199.99") }
{ "_id" : 2, "title" : "Melancholy III", "artist" : "Munch", "year" : 1902,
    "price" : NumberDecimal("280.00") }
{ "_id" : 3, "title" : "Dancer", "artist" : "Miro", "year" : 1925,
    "price" : NumberDecimal("76.04") }
{ "_id" : 4, "title" : "The Great Wave off Kanagawa", "artist" : "Hokusai",
    "price" : NumberDecimal("167.30") }
{ "_id" : 5, "title" : "The Persistence of Memory", "artist" : "Dali", "year" : 1931,
    "price" : NumberDecimal("483.00") }
{ "_id" : 6, "title" : "Composition VII", "artist" : "Kandinsky", "year" : 1913,
    "price" : NumberDecimal("385.00") }
{ "_id" : 7, "title" : "The Scream", "artist" : "Munch", "year" : 1893
    /* No price*/ }
{ "_id" : 8, "title" : "Blue Flower", "artist" : "O'Keefe", "year" : 1918,
    "price" : NumberDecimal("118.42") }
```

$bucket

Categorizes incoming documents into groups, called buckets, based on a specified expression and bucket boundaries.

```
db.artwork.aggregate( [
  {
    $bucket: {
      groupBy: "$price",
      boundaries: [ 0, 200, 400 ],
      default: "Other",
      output: {
        "count": { $sum: 1 },
        "titles" : { $push: "$title" }
      }
    }
  }
] )
```



```
{
  "_id" : 0,
  "count" : 4,
  "titles" : [
    "The Pillars of Society",
    "Dancer",
    "The Great Wave off Kanagawa",
    "Blue Flower"
  ]
}
{
  "_id" : 200,
  "count" : 2,
  "titles" : [
    "Melancholy III",
    "Composition VII"
  ]
}
{
  "_id" : "Other",
  "count" : 2,
  "titles" : [
    "The Persistence of Memory",
    "The Scream"
  ]
}
```

$bucketAuto

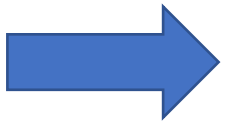
Categorizes incoming documents into a specific number of groups, called buckets, based on a specified expression. Bucket boundaries are automatically determined in an attempt to evenly distribute the documents into the specified number of buckets.

```
db.artwork.aggregate( [
    {
      $bucketAuto: {
          groupBy: "$price",
          buckets: 4
      }
    }
] )
```

```
{
  "_id" : {
    "min" : NumberDecimal("76.04"),
    "max" : NumberDecimal("159.00")
  },
  "count" : 2
}
{
  "_id" : {
    "min" : NumberDecimal("159.00"),
    "max" : NumberDecimal("199.99")
  },
  "count" : 2
}
{
  "_id" : {
    "min" : NumberDecimal("199.99"),
    "max" : NumberDecimal("385.00")
  },
  "count" : 2
}
{
  "_id" : {
    "min" : NumberDecimal("385.00"),
    "max" : NumberDecimal("483.00")
  },
  "count" : 2
}
```

$count Returns a count of the number of documents at this stage of the aggregation pipeline.

```
{ "_id" : 1, "subject" : "History", "score" : 88 }
{ "_id" : 2, "subject" : "History", "score" : 92 }
{ "_id" : 3, "subject" : "History", "score" : 97 }
{ "_id" : 4, "subject" : "History", "score" : 71 }
{ "_id" : 5, "subject" : "History", "score" : 79 }
{ "_id" : 6, "subject" : "History", "score" : 83 }
```



```
db.scores.aggregate(
  [
    {
      $match: {
        score: {
          $gt: 80
        }
      }
    },
    {
      $count: "passing_scores"
    }
  ]
)
```

```
{ "passing_scores" : 4 }
```

```
db.scores.aggregate(
[
        { $match: { extraCredit: { $gt: 0 } } },
        { $count: "passing_extraCredits" }
]
)
```

{ "passing_extraCredits" : 1 }

`$facet`	Processes multiple aggregation pipelines within a single stage on the same set of input documents. Enables the creation of multi-faceted aggregations capable of characterizing data across multiple dimensions, or facets, in a single stage.

```
{ $facet:
   {
      <outputField1>: [ <stage1>, <stage2>, ... ],
      <outputField2>: [ <stage1>, <stage2>, ... ],
      ...

   }
}
```

`$facet`

Processes multiple aggregation pipelines within a single stage on the same set of input documents. Enables the creation of multi-faceted aggregations capable of characterizing data across multiple dimensions, or facets, in a single stage.

```
{ "_id" : 1, "title" : "The Pillars of Society", "artist" : "Grosz", "year" : 1926,
  "price" : NumberDecimal("199.99"),
  "tags" : [ "painting", "satire", "Expressionism", "caricature" ] }
{ "_id" : 2, "title" : "Melancholy III", "artist" : "Munch", "year" : 1902,
  "price" : NumberDecimal("280.00"),
  "tags" : [ "woodcut", "Expressionism" ] }
{ "_id" : 3, "title" : "Dancer", "artist" : "Miro", "year" : 1925,
  "price" : NumberDecimal("76.04"),
  "tags" : [ "oil", "Surrealism", "painting" ] }
{ "_id" : 4, "title" : "The Great Wave off Kanagawa", "artist" : "Hokusai",
  "price" : NumberDecimal("167.30"),
  "tags" : [ "woodblock", "ukiyo-e" ] }
{ "_id" : 5, "title" : "The Persistence of Memory", "artist" : "Dali", "year" : 1931,
  "price" : NumberDecimal("483.00"),
  "tags" : [ "Surrealism", "painting", "oil" ] }
{ "_id" : 6, "title" : "Composition VII", "artist" : "Kandinsky", "year" : 1913,
  "price" : NumberDecimal("385.00"),
  "tags" : [ "oil", "painting", "abstract" ] }
{ "_id" : 7, "title" : "The Scream", "artist" : "Munch", "year" : 1893,
  "tags" : [ "Expressionism", "painting", "oil" ] }
{ "_id" : 8, "title" : "Blue Flower", "artist" : "O'Keefe", "year" : 1918,
  "price" : NumberDecimal("118.42"),
  "tags" : [ "abstract", "painting" ] }
```

```
db.artwork.aggregate( [
  {
    $facet: {
      "categorizedByTags": [
        { $unwind: "$tags" },
        { $sortByCount: "$tags" }
      ],
      "categorizedByPrice": [
        // Filter out documents without a price e.g., _id: 7
        { $match: { price: { $exists: 1 } } },
        {
          $bucket: {
            groupBy: "$price",
            boundaries: [  0, 150, 200, 300, 400 ],
            default: "Other",
            output: {
              "count": { $sum: 1 },
              "titles": { $push: "$title" }
            }
          }
        }
      ],
      "categorizedByYears(Auto)": [
        {
          $bucketAuto: {
            groupBy: "$year",
            buckets: 4
          }
        }
      ]
    }
  }
```

`$facet`	Processes multiple aggregation pipelines within a single stage on the same set of input documents. Enables the creation of multi-faceted aggregations capable of characterizing data across multiple dimensions, or facets, in a single stage.

```
pipe1stage1 ={ $unwind: "$tags" }
pipe1stage2 ={ $sortByCount: "$tags" }
pipe2stage1 = { $match: { price: { $exists: 1 } } }
pipe2stage2 = { $bucket: { groupBy: "$price", boundaries: [ 0, 150, 200, 300, 400 ], default: "Other", output: { "count": { $sum: 1 },  "titles": { $push: "$title" } }}}}
pipe3stage1= { $bucketAuto: { groupBy: "$year", buckets: 4 } }
pipe1 = [pipe1stage1,pipe1stage2]
pipe2 =[pipe2stage1,pipe2stage2]
pipe3 =[pipe3stage1]
faceta ={"categorizedByTags":pipe1,"categorizedByPrice":pipe2,"categorizedByYears(Auto)":pipe3}
db.artwork.aggregate([ {$facet:faceta}])
```

$group	Groups input documents by a specified identifier expression and applies the accumulator expression(s), if specified, to each group. Consumes all input documents and outputs one document per each distinct group. The output documents only contain the identifier field and, if specified, accumulated fields.

```
{ $group: { _id: <expression>, <field1>: { <accumulator1> : <expression1> }, ... } }
```

- Agrupa los documentos acorde con alguna expresión especificada y genera un documento para cada agrupamiento diferente
- Los documentos de salida contienen un campo _id el cual contiene el grupo distinto por llave-
- Los documentos de salida pueden también contener campos computados que tienen los valores de alguna expresión de acumulador agrupadas por el campo _id de $group.
- $group no ordena los documentos de salida.
- El campo _id es obligafo, mientras que el resto de los campos son opcionales, que se calculan con operadores <accumulator>

Los operadores <accumulator>

`$avg`	Returns an average of numerical values. Ignores non-numeric values.
`$first`	Returns a value from the first document for each group. Order is only defined if the documents are in a defined order.
`$last`	Returns a value from the last document for each group. Order is only defined if the documents are in a defined order.
`$max`	Returns the highest expression value for each group.
`$min`	Returns the lowest expression value for each group.

Los operadores <accumulator>

`$push`	Returns an array of expression values for each group.
`$addToSet`	Returns an array of *unique* expression values for each group. Order of the array elements is undefined.
`$stdDevPop`	Returns the population standard deviation of the input values.
`$stdDevSamp`	Returns the sample standard deviation of the input values.
`$sum`	Returns a sum of numerical values. Ignores non-numeric values.

Ejemplo: la colección sales

{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }

Grupos por mes, por día y por año

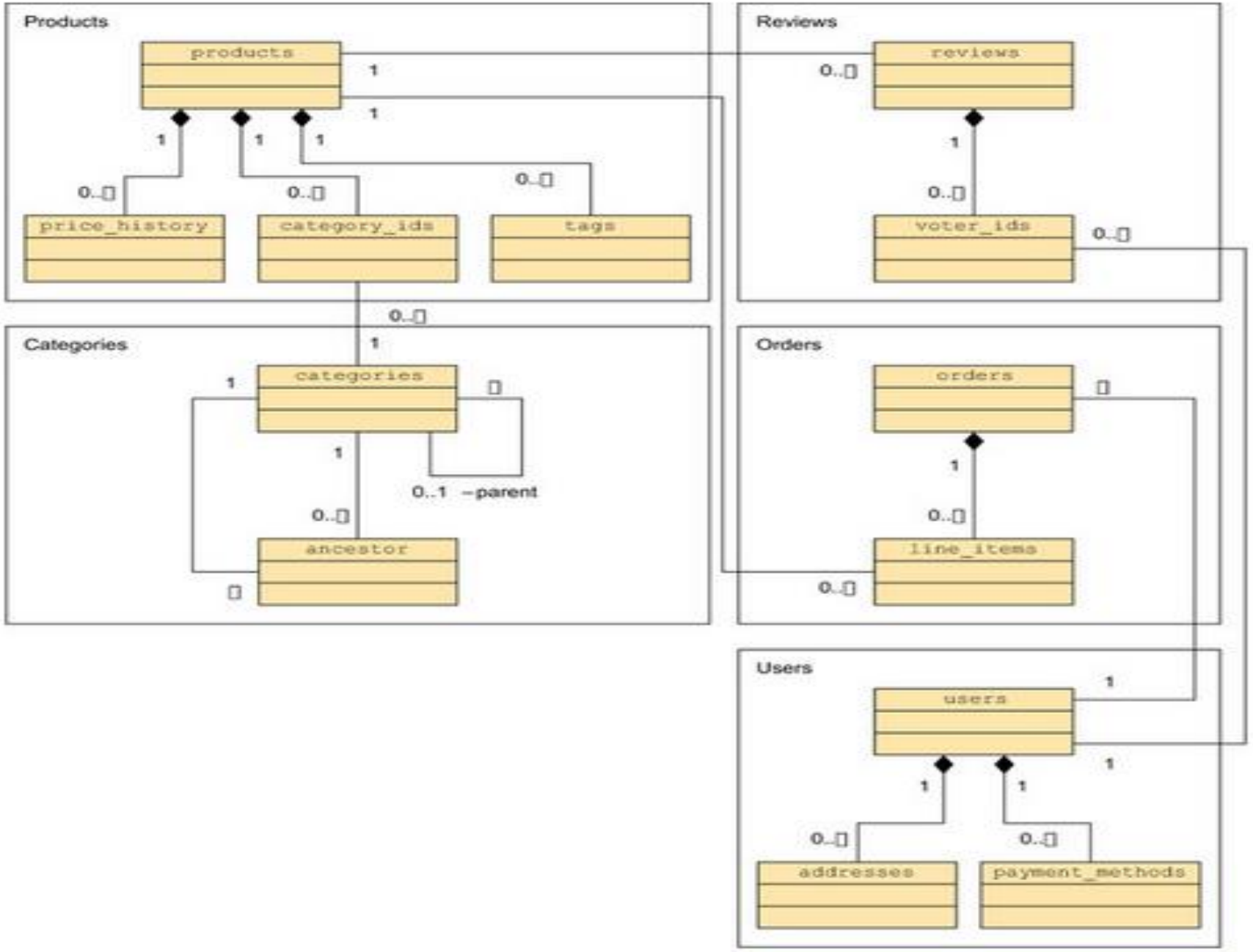
Usamos la etapa $group para agrupar los documentos por mes, día y año y calculamos el precio total y su promedi, así como la cantidad de documentos para cada grupo

```
db.sales.aggregate(
  [
    {
      $group : { _id : { month: { $month: "$date" }, day: { $dayOfMonth: "$date" }, year: {
$year: "$date" } },
          totalPrice: { $sum: { $multiply: [ "$price", "$quantity" ] } },
          averageQuantity: { $avg: "$quantity" },
          count: { $sum: 1 }
      }
    }
  ]
)
```

Creando variables

```
grouping = { month: { $month: "$date" }, day: { $dayOfMonth: "$date" }, year: { $year: "$date" } }
precioTotal=  { $sum: { $multiply: [ "$price", "$quantity" ] } }
promedio =    { $avg: "$quantity" }
cantidad =    { $sum: 1 }

db.sales.aggregate([ {$group: {_id:                grouping,
                               totalPrice:         precioTotal,
                               averageQuantity: promedio,
                               count:               cantidad
                              }
                     }
                   ]
)
```

Operaciones de agregación con la base de datos ecommerce



Contar el número de revisiones para cada producto

```
product  = db.products.findOne({'slug': 'wheelbarrow-9092'})
reviews_count = db.reviews.count({'product_id': product['_id']})
```

```
db.reviews.aggregate([
 {$group : { _id:'$product_id',
             count:{$sum:1} }}
]);
```

Group the input documents by product_id.

Count the number of reviews for each product.

La operación de agregación map-reduce

MapReduce

- MapReduce es un paradigma de procesamiento de datos para condensar grandes volúmenes de datos en resultados agregados útiles.
- MapReduce puede resolver algunos problemas que pueden ser muy complejos para expresar utilizando agregación.
- MapReduce utiliza JavaScript

Otra manera de realizar operaciones de agregación

Mapear y reducir

MongoDb brinda las operaciones de **`map-reduce`** para realizar procesos de agregación.

Dos fases:

1. Una fase **`map`** que procesa cada documento y emite uno o más objetos para cada documento de entrada.
2. Una fase **`reduce`** que combina la salida de la primera fase **`map`**

De forma opcional, la operación **`map-reduce`** tiene la fase **`finalize`** que permite realizar modificaciones finales al resultado.

Al igual que las operaciones del marco de trabajo de agregación, **`map-reduce`** puede especificar una condición de consulta para seleccionar los documentos de entrada, así como poder ordenarlos y limitar los resultados.

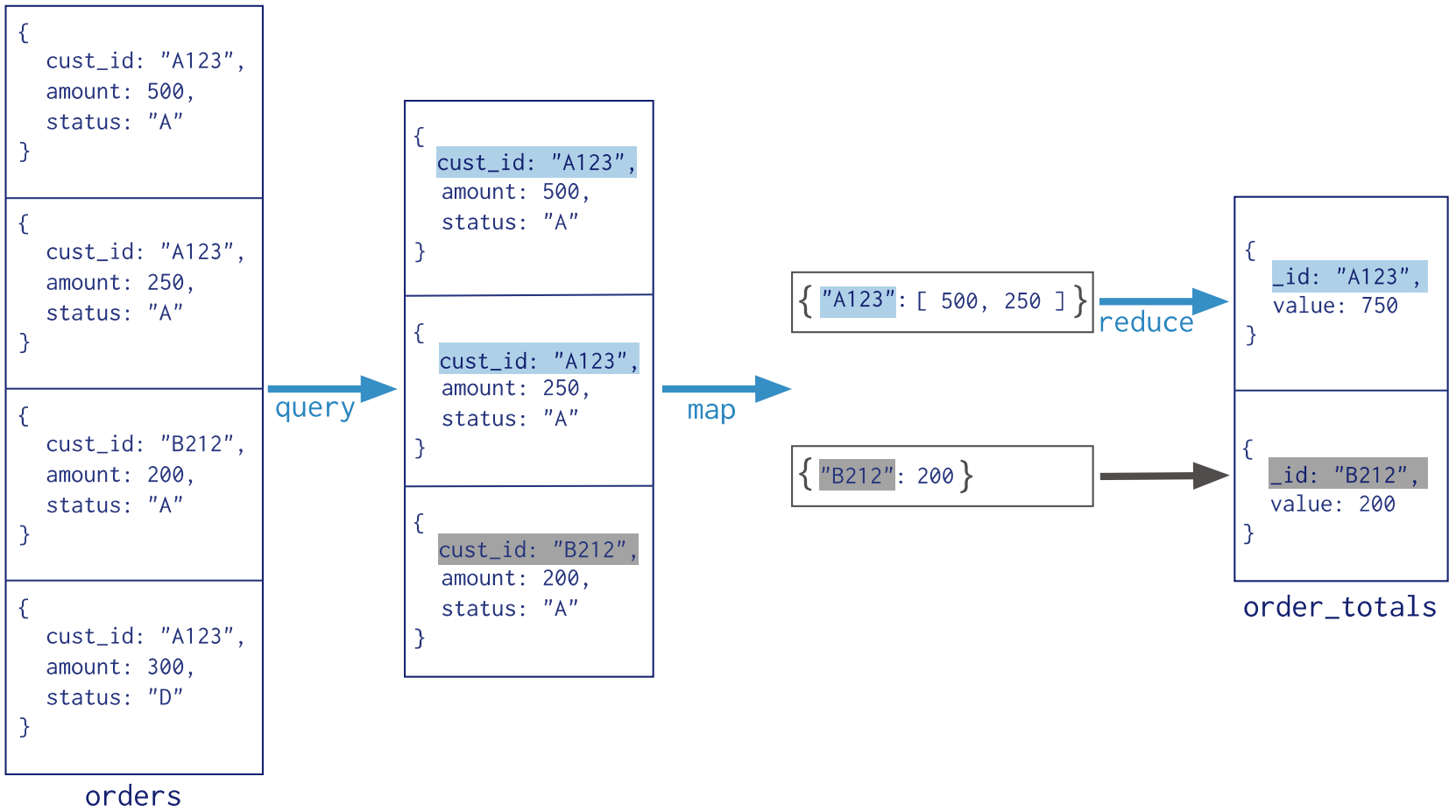
`map-reduce` utiliza funciones customs de Javascript para llevar a cabo las operaciones de mapear y reducir, así como también las operaciones de la fase **`finalize`**.

¿Qué usar , **`map-reduce`** o el marco de trabajo de agregación?

map-reduce

Collection

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



MapReduce

- Map
  - Mapea una operación en cada documento en la colección
  - Puede ser que no haga nada o emita estas llaves con valores

  - Etapa intermedia (shuffle step)
    - Las llaves son agrupadas y se crea una lista de los valores emitidos para cada llave
- Reduce
  - Toma la lista de los valores y los reduce a un solo elemento. Este elemento es retornado a la etapa intermedia (shuffle step) hasta que cada llave de la lista contenga un solo valor: el resultado

Sintaxis

- **map** es una función de JavaScript que mapea un valor con una clave y emite un par llave-valor
- **reduce** es una función de JavaScript que reduce o agrupa todos los documentos que tienen la misma llave
- **out** especifica la ubicación del resultado de la consulta map-reduce
- **query** especifica los criterios de selección opcionales para seleccionar documentos
- **sort** especifica los criterios de ordenamiento opcionales
- **limit** especifica la cantidad máxima opcional de documentos a devolver

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

Ejemplo 1

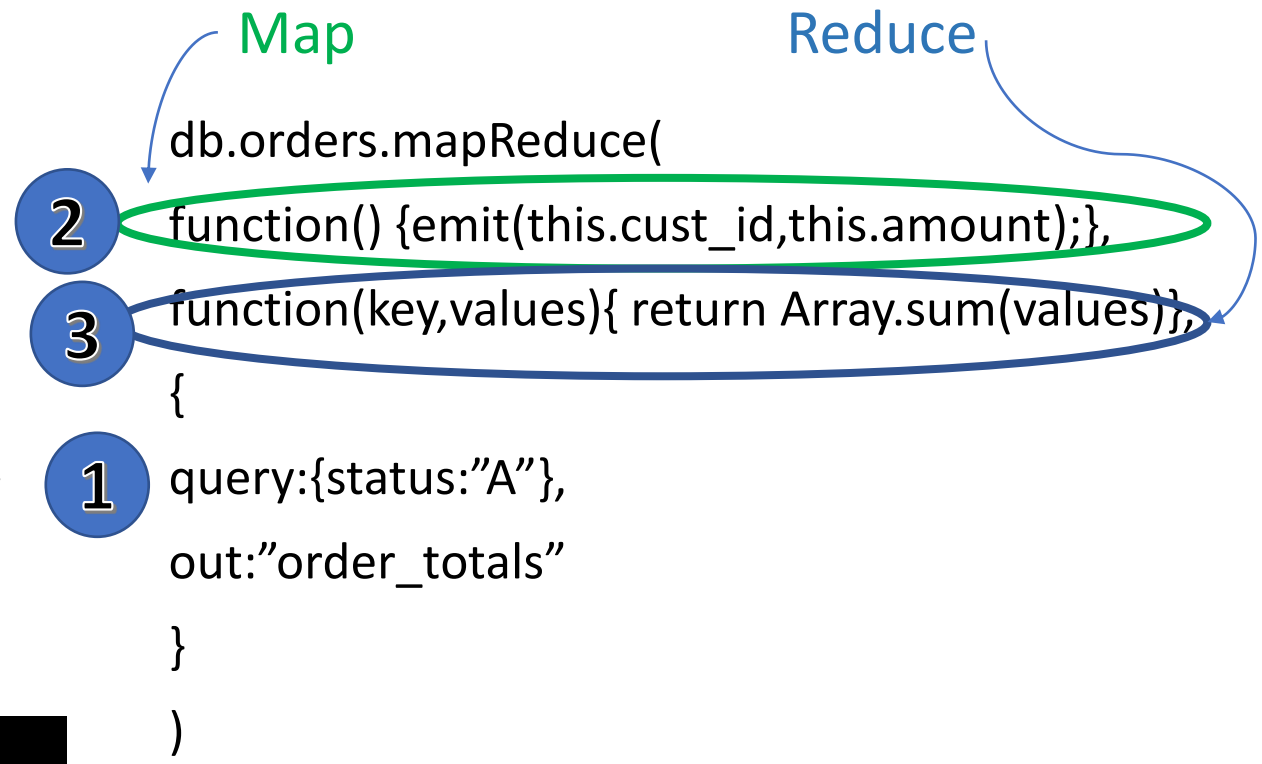
- Dados la siguiente colección (orders)
- Deseamos obtener las cantidades totales (amount) por cliente (cust_id), cuyo status sea “A”

```
use prueba
d1={cust_id:"A123",amount:500,status:"A"}
d2={cust_id:"A123",amount:250,status:"A"}
d3={cust_id:"B212",amount:200,status:"A"}
d4={cust_id:"A123",amount:300,status:"D"}
db.orders.insertMany([d1,d2,d3,d4])
```

Ejemplo 1

```
use prueba
d1={cust_id:"A123",amount:500,status:"A"}
d2={cust_id:"A123",amount:250,status:"A"}
d3={cust_id:"B212",amount:200,status:"A"}
d4={cust_id:"A123",amount:300,status:"D"}
db.orders.insertMany([d1,d2,d3,d4])
```

```
> db.order_totals.find()
{ "_id" : "A123", "value" : 750 }
{ "_id" : "B212", "value" : 200 }
```



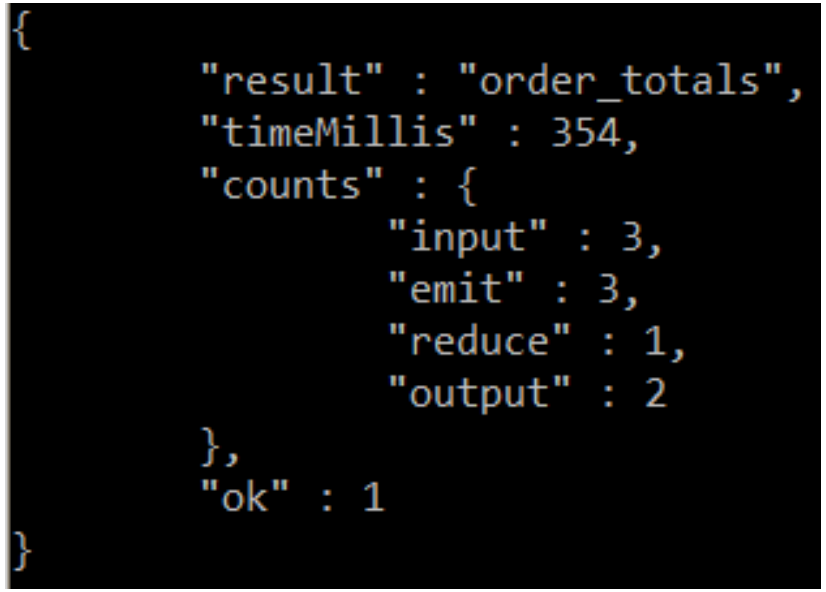
Ejemplo 1 (cont.)

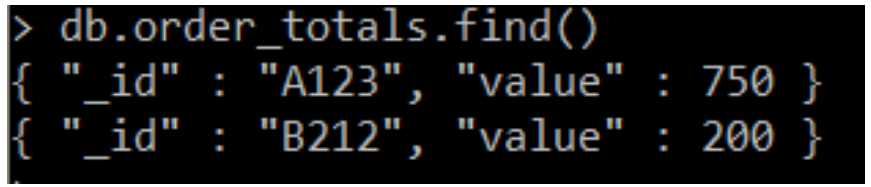
- Función map
  - This se refiere al documento que la operación de map-reduce está procesando
  - La función mapea price y cust_id para cada documento y emite un par cust_id y price (amount)
- Función reduce
  - Recibe los pares llave-valor y obtiene las sumas de amount

```
db.orders.mapReduce(
function() {emit(this.cust_id,this.amount);},
function(key,values){ return Array.sum(values)},
{
query:{status:”A”},
out:”order_totals”
}
)
```

Ejemplo 1 (cont.)

- timeMills. Tiempo que llevó la operación en milisegundos
- counts: {…}, este documento embebido contiene cuatro llaves:
  - input: número de documentos enviados a la función map
  - emit: número de veces que emit es llamado en la función map
  - output: número de documentos creados en la colección
  - reduce: número de veces que redujo los valores

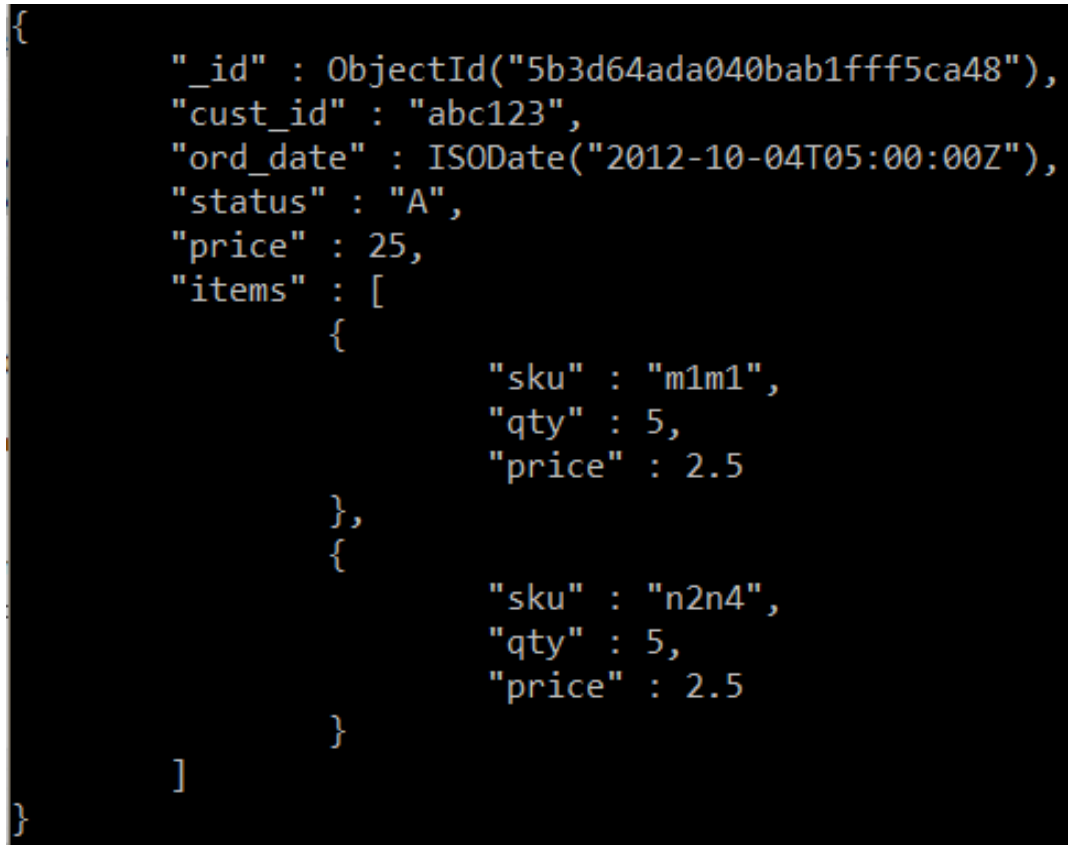




Ejemplo 2

Queremos obtener el total (price) por cada cliente (cust_id)

```
use prueba
d1={
 cust_id: "abc123",
    ord_date: new Date("Oct 04, 2012"),
    status: 'A',
    price: 25,
    items: [ { sku: “m1m1", qty: 5, price: 2.5 },
           { sku: "n2n4", qty: 5, price: 2.5 } ]
}
db.item.insert(d1)
```



Ejemplo 2 (cont.)

1

```
var mapFun=function(){
    emit(this.cust_id, this.price);
};
```

key	value
abc123	25

```
db.item.mapReduce(
                    mapFun,
                    reduceFun,
                    { out: “map_red_01}
)
```

2

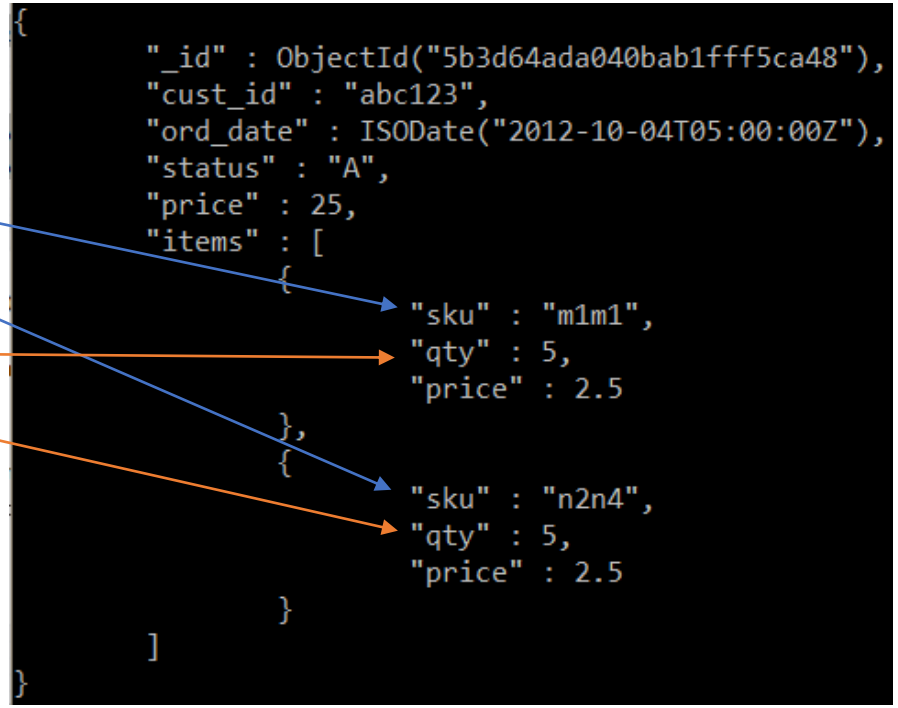
```
var reduceFun=
    function(keyCust,valuePrice){
    return Array.sum(valuePrice);
};
```

```
> db.item.mapReduce( mapFun,  reduceFun, { out: "map_red_01"} )
{
        "result" : "map_red_01",
        "timeMillis" : 433,
        "counts" : {
                "input" : 1,
                "emit" : 1,
                "reduce" : 0,
                "output" : 1
        },
        "ok" : 1
}
> db.map_red_01.find()
{ "_id" : "abc123", "value" : 25 }
```

Ejemplo 3

- Queremos obtener las cantidades (qty) totales por elemento (sku), y la cantidad promedio por orden de cada elemento (calculado como el número de órdenes entre la cantidad total ordenada por cada elemento (sku)), para aquellos documentos con fecha de orden mayor al 01/01/2012.

```
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);
    }
};
```



key	value
m1m1	count:1, qt1:5
n2n4	count:1, qty:5

Ejemplo 3 (cont.)

```
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);
                  }
                };

 var reduceFunction2 = function(keySKU, countObjVals) {
                  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;
                };
```

key	value
m1m1	count:1, qty:5
n2n4	count:1, qty:5


```
{
        "result" : "map_reduce_example",
        "timeMillis" : 405,
        "counts" : {
                "input" : 1,
                "emit" : 2,
                "reduce" : 0,
                "output" : 2
        },
        "ok" : 1
}
```

Ejemplo 3 (cont.)

- finalizeFunction modifica el objeto reduceVal para agregar un campo calculado llamado avg.

```
var finalizeFunction2 = function (key, reducedVal) {

        reducedVal.avg = reducedVal.qty/reducedVal.count;

        return reducedVal;

    };
```

```
> db.map_reduce_example.find()
{ "_id" : "m1m1", "value" : { "count" : 1, "qty" : 5, "avg" : 5 } }
{ "_id" : "n2n4", "value" : { "count" : 1, "qty" : 5, "avg" : 5 } }
```

Ejemplo 4

Calcular el número de cursos por profesor (obtenga solo los cursos de Alice Jones)

```
db.classes.insert({
class : "Biology 101",
startDate : new Date(2016, 1, 11),
students : [
    {fName : "Andy", lName : "Brennan", age : 36},
    {fName : "James", lName : "Hurley", age : 25},
    {fName : "Harry", lName : "Truman", age : 41}
],
cost : 1550,
professor : "Alice Jones",
topics : "Earth,Cell,Energy,Genetics,DNA",
book:
{
    isbn: "0547219474",
    title: "Holt McDougal Biology",
    price: 104.30
}
})

db.classes.insert({
class : "Chemistry 101",
startDate : new Date(2016, 1, 13),
students : [
    {fName : "Bobby", lName : "Briggs", age : 21},
    {fName : "Donna", lName : "Hayward", age : 21},
    {fName : "Audrey", lName : "Horne", age : 20},
    {fName : "James", lName : "Hurley", age : 25}
],
cost : 1600,
professor : "Alice Jones",
topics : "Matter,Energy,Atom,Periodic Table",
book:
{
    isbn: "0547219474",
    title: "Chemistry : Matter and Change",
    price: 104.30
}
})
```

2 var mapFunc2=function(){

emit(this.professor,1);}

3 var redFunc2=function(pro,cta){

return Array.sum(cta);}

1 db.clases.mapReduce(mapFunc2,redFunc2,

{query:{professor:"Alice Jones"},

out: "map_ex_2"})

```
> db.map_ex_2.find()
{ "_id" : "Alice Jones", "value" : 3 }
```

Ejercicio

- Obtener el número de clases que tiene cada alumno, genere la llave de tal modo que este conformada por el nombre, espacio en blanco y apellido

Ejercicio (cont.)

```
var mapFunction3 = function() {
for (var idx = 0; idx < this.students.length; idx++) {
    var key = this.students[idx].fName+" "+this.students[idx].lName;
    emit(key, 1); }};

var reduceFunction3 = function(key, countSt) {
reducedVal = 0;
for (var idx = 0; idx < countSt.length; idx++) {
    reducedVal+=countSt[idx]; }
return reducedVal;};

db.classes.mapReduce( mapFunction3, reduceFunction3, { out: "map_red_031"})
```

```
> db.map_red_031.find()
{ "_id" : "Andy Brennan", "value" : 2 }
{ "_id" : "Audrey Horne", "value" : 3 }
{ "_id" : "Bobby Briggs", "value" : 2 }
{ "_id" : "Dale Cooper", "value" : 2 }
{ "_id" : "Donna Hayward", "value" : 2 }
{ "_id" : "Harry Truman", "value" : 1 }
{ "_id" : "James Hurley", "value" : 2 }
{ "_id" : "Laura Palmer", "value" : 3 }
{ "_id" : "Lucy Moran", "value" : 1 }
{ "_id" : "Shelly Johnson", "value" : 1 }
{ "_id" : "Tommy Hill", "value" : 1 }
```

```
> db.classes.mapReduce( mapFunction3,  reduceFunction3, { out: "map_red_031"} )
{
        "result" : "map_red_031",
        "timeMillis" : 408,
        "counts" : {
                "input" : 6,
                "emit" : 20,
                "reduce" : 7,
                "output" : 11
        },
        "ok" : 1
}
```

Map-Reduce en BDD Garden

Map-Reduce

2 Map

```
map = function() {
   var shipping_month = (this.purchase_data.getMonth()+1) +
      '-' + this.purchase_data.getFullYear();

   var tmpItems = 0;
   this.line_items.forEach(function(item) {
      tmpItems += item.quantity;
   });

   emit(shipping_month, {order_total: this.sub_total,
                         items_total: tmpItems});
};
```

3 Reduce

```
reduce = function(key, values) {
   var result = { order_total: 0, items_total: 0 };
   values.forEach(function(value){
      result.order_total += value.order_total;
      result.items_total += value.items_total;
   });
   return ( result );
};
```

1

```
filter = {purchase_data: {$gte: new Date(2010, 0, 1)}};
db.orders.mapReduce(map, reduce, {query: filter, out: 'totals'});
```

2

Map

Construye la llave con el mes y el año de compra separado por "-"

```
map = function() {
    var shipping_month = (this.purchase_data.getMonth()+1) +
        '-' + this.purchase_data.getFullYear();

    var tmpItems = 0;
    this.line_items.forEach(function(item) {
        tmpItems += item.quantity;
    });

    emit(shipping_month, {order_total: this.sub_total,
                          items_total: tmpItems});
};
```

Cuenta los productos que hay en esa orden

Par llave-valor


```
"user_id" : ObjectId("4c4b1476238d3b4dd5000001"),
"purchase_data" : ISODate("2014-08-01T07:00:00Z"),
"state" : "CART",
"line_items" : [
        {
                "_id" : ObjectId("4c4b1476238d3b4dd5003981"),
                "sku" : "9092",
                "name" : "Extra Large Wheel Barrow",
                "quantity" : 1,
                "pricing" : {
                        "retail" : 5897,
                        "sale" : 4897
                }
        },
        {
                "_id" : ObjectId("4c4b1476238d3b4dd5003982"),
                "sku" : "10027",
                "name" : "Rubberized Work Glove, Black",
                "quantity" : 1,
                "pricing" : {
                        "retail" : 1499,
                        "sale" : 1299
                }
        }
],
"shipping_address" : {
        "street" : "588 5th Street",
        "city" : "Brooklyn",
        "state" : "NY",
        "zip" : 11215
},
"sub_total" : 6196,
"tax" : 600
```

3 Reduce

key	value
11-2004	[4897,1]
4-2014	[4897,1]
8-2014	[6196,2] [4897,1]

key	value
11-2004	[4897,1]
4-2014	[4897,1]
8-2014	[11093,3]

- Colección totals

```
> db.orders.mapReduce(map, reduce, {query: filter, out: 'totals'});
{
        "result" : "totals",
        "timeMillis" : 605,
        "counts" : {
                "input" : 4,
                "emit" : 4,
                "reduce" : 1,
                "output" : 3
        },
        "ok" : 1
}
```

```
> db.totals.find()
{ "_id" : "11-2014", "value" : { "order_total" : 4897, "items_total" : 1 } }
{ "_id" : "4-2014", "value" : { "order_total" : 4897, "items_total" : 1 } }
{ "_id" : "8-2014", "value" : { "order_total" : 11093, "items_total" : 3 } }
```