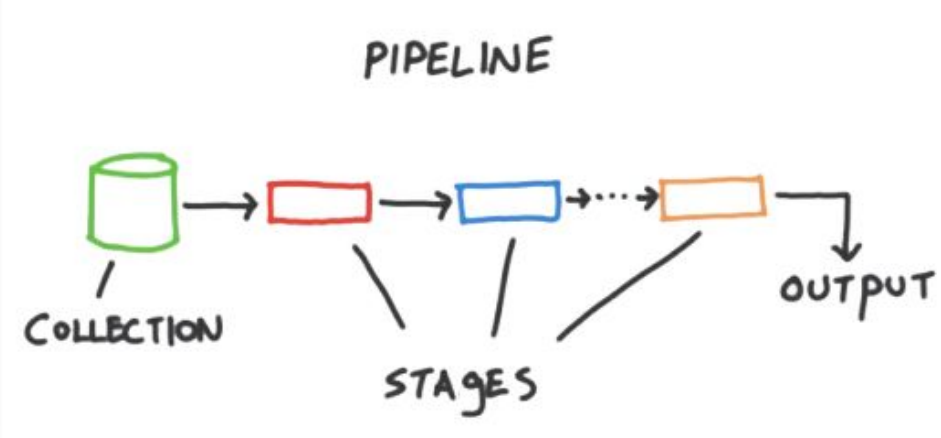
# Bases de No Datos Relacionales

- Framework de agregación

## ¿Qué es el Framework de agregación?

Es una herramienta que permite procesar grandes cantidades de datos de manera eficiente. En lugar de recuperar documentos completos mediante una consulta y luego procesarlos en la aplicación, se puede definir una serie (*pipeline*) de etapas (*stages*) para **transformar y resumir** los datos dentro de MongoDB, para luego enviar como salida a la aplicación el resultado.



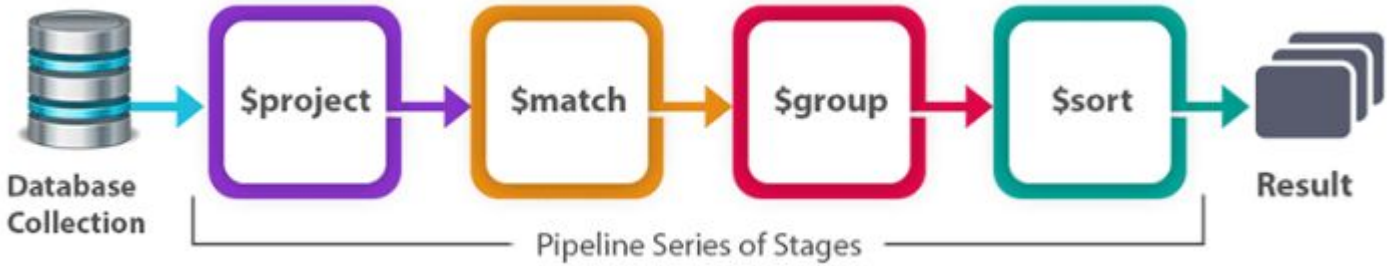
## ¿Cómo funciona el framework de agregación?

Una canalización de agregación o pipeline consta de una o varios stages:

- Cada etapa realiza una operación en los documentos de entrada. Por ejemplo, una fase puede filtrar documentos, agruparlos, calcular valores.
- Los documentos que se generan de una fase se pasan a la siguiente etapa.
- Un pipeline puede devolver resultados para un grupo de documentos. Por ejemplo, devuelve los valores total, promedio, máximo y mínimo.

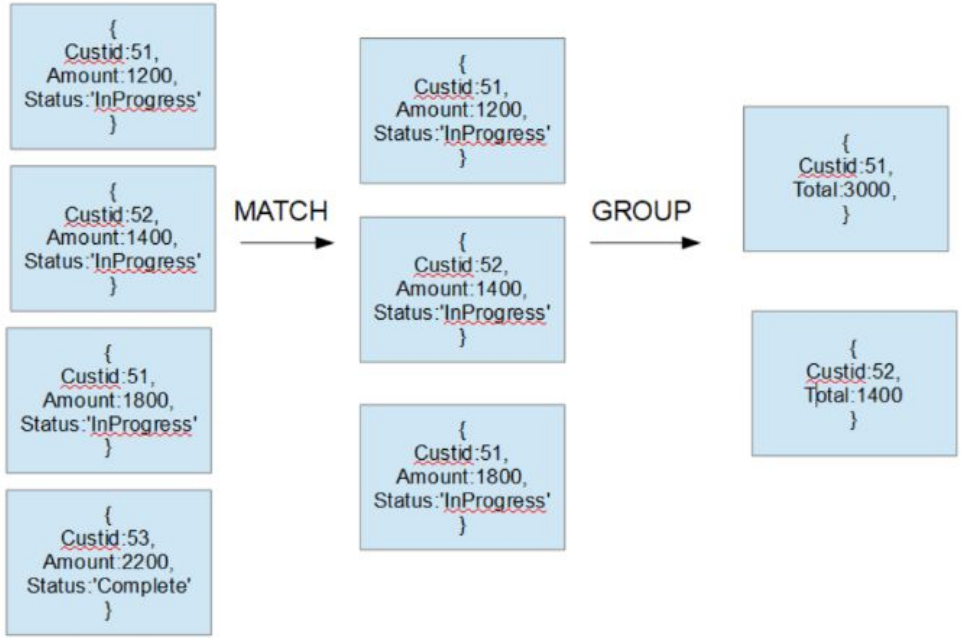
## ¿Qué es el Framework de agregación?

Aquí vemos un ejemplo concreto de algunos posibles stages de mongo combinados para formar un pipeline que produce al final un resultado.



## ¿Qué es el Framework de agregación?

Supongamos que de la siguiente colección, queremos obtener por *Custid*, el total de *Amount* de los procesos con *status* “InProgress”



## Paralelismo con el lenguaje SQL

| SQL | MongoDB Aggregation Operators |
|---|---|
| SELECT | $project |
| LIMIT | $limit |
| WHERE | $match |
| GROUP BY | $group |
| HAVING | $match |
| ORDER BY | $sort |
| SUM() | $sum |
| COUNT() | $sum |
| Join | $lookup (para colecciones distintas)<br>$unwind (para arrays de una misma colección) |

- **Sintaxis**

```
db.collection_name.aggregate ([
                        {$pipeline_operator: “expression”},
                        {$pipeline_operator: {document_expression}},
                            … ])
```

**Veamos un ejemplo para identificar stages y operadores:**

```
db.pizzas.aggregate( [

// Stage 1: Filtra los documentos por el tamaño de las pizzas
{$match: { size: "medium" }}
,
// Stage 2: Agrupa los documentos por el tipo de pizza y suma las ventas
{$group: { _id: "$type", TOTAL: { $sum: "$stats.sales" }}}
                    ] )
```

**La salida se verá similar a esto**

```
[
      { _id: 'Cheese', TOTAL: 50 },
      {  id: 'Vegan', TOTAL: 10 },
      { _id: 'Pepperoni', TOTAL: 20 }
]
```

- **Veamos un poco más en detalle cada operador**

  - **$group**

    Los grupos se arman especificando los atributos en el _id.
    Los operadores disponibles son los siguientes:

    $sum, $avg, $max, $min, $first, $last, $push, $addToSet

    Ejemplo : { $group: { _id: , : { : }, ... } }

  - **$match** En esta operación se pueden usar algunos de los operadores ya vistos con el find:

    $eq, $gt, $gte, $lt, $lte, $ne, $in, $nin, $or, $and, $not, $exists, $type, $regex, $text, $all, $elemMatch, $size

- **Veamos un poco más en detalle cada operador**

- **$project** Se realiza una proyección de los documentos. Pueden seleccionarse sierts atributos, renombrarlos y realizar operaciones. En ocasiones es beneficioso hacer un project antes que otra operación para reducir la cantidad de atributos a tratar, especialmente en colecciones grandes o cuando trabajamos con grandes arrays.

  **Renombrar un atributo:** Se deberá poner $ antes del atributo.
  {apellido:"$cliente.apellido"}

  **Operaciones matemáticas:**
  {$add:[, …, ]} {$multiply:[, …, ]} {$substract:[, ]} {$divide:[, ]}

  Ej: total: {$multiply:["$stats.sales", "$stats.price"]}

- **Veamos un poco más en detalle cada operador**

  - **$project** (...continuación)

    **Operaciones con strings:**

    {$concat:[, …, ]} {$toLower:} {$toUpper:} {$substr:[, , ]}

    **Operaciones con fechas:**

    $year, $month, $week, $dayOfMonth, $dayOfWeek, $dayOfYear, $hour, $minute

    Ej: {año:{$substract:[{$year: new Date()},{$year: $fechaInicial}]}}

- **Veamos un poco más en detalle cada operador**

  - **$project** (...continuación)

    **Operaciones de control:**
    {$ifNull:[, ]} {$cond:[, , ]} ó {$cond:{if: , then: , else: }}

    Ej: descuento:{$cond:[{$gte:["$cant",10]},20,5]}

    **Operaciones con Arrays:**

    {$arrayElemAt:[, ]}
    {$map:{input:{expresion}, as:"nombre", in:]}
    {$setIntersection:{, , …}}
    {$filter:{input:, as:"nombre", condBooleana: } }

- **Ejemplos de uso en comparación con sentencias SQL**

Supongamos que trabajamos con la siguiente colección Facturas cuyos documentos tienen la siguiente estructura en general:

```
{ "_id" : ObjectId("536183a934600053a7b6bc67"),
"nroFactura" : 1466,
"fechaEmision" : ISODate("2014-04-29T00:00:00Z"),
"condPago" : "CONTADO",
"cliente" : "GALINDO S.A.",
"dni" : 30432481,
"region" : "CABA",
"totalFactura" : 1500,
"item" :      [
              { "producto" : "mesa 2 x 1 m", "cantidad" : 1 },
              { "producto" : "sillas Z322", "cantidad" : 4 }
              ]
 }
```

## Ejemplos de uso en comparación con sentencias SQL

**SQL:**

SELECT COUNT(*) AS Cuenta FROM facturas

**Mongo:**

```
db.facturas.aggregate( [
                        {
                         $group: { _id: null, Cuenta: { $sum: 1 } }
                         }
                    ])
```

**Retorna la cantidad de facturas de la colección y la presenta bajo el atributo Cuenta, el resultado se verá así:**

```
{
  _id: null,
  Cuenta: XXXX
}
```

## Ejemplos de uso en comparación con sentencias SQL

**SQL:**

```
SELECT SUM(totalFactura) AS total FROM facturas
```

**Mongo:**

```
db.facturas.aggregate([
                        {
                         $group: { _id: null, total : { $sum: "$totalFactura" }}
                         }
                    ])
```

**Retorna el monto total de todas las facturas de la colección y lo presenta bajo el atributo total, el resultado se verá así:**

```
{
  _id: null,
  total : xxxx
}
```

## Ejemplos de uso en comparación con sentencias SQL

**SQL:** SELECT condPago, SUM(totalFactura) AS total FROM facturas GROUP BY condPago

**Mongo:**
```
db.facturas.aggregate([
                        {
                         $group: { _id: "$condPago", total : { $sum: "$totalFactura" }}
                         }
                    ])
```

**Retorna el monto total de todas las facturas de la colección agrupadas por la condición de pago**

**el resultado se verá así:**

```
{
  _id: 'DEBITO',
  total: XXXXX
}
.....
{
  _id: 'CREDITO',
  total: ZZZZ
}
```

## Ejemplos de uso en comparación con sentencias SQL

**SQL:**

```
SELECT condPago, SUM(totalFactura) AS total FROM facturas
GROUP BY condPago SORT BY total
```

**Mongo:**

```
db.facturas.aggregate([
                    {
                    $group:
                        {       _id: "$condPago" ,
                                total: { $sum: "$totalFactura" }
                        }
                    } ,
                    {$sort: { total:1}
                    ])
```

Retorna el monto total de todas las facturas de la colección agrupadas por la condición de pago

el resultado se verá igual que antes solo que ordenado de mayor a menor por el cálculo realizado

## Ejemplos de uso en comparación con sentencias SQL

**SQL:** SELECT cliente, condPago, SUM(totalFactura) AS total FROM facturas
GROUP BY cliente, condPago

**Mongo:**

```
db.facturas.aggregate([
                    {
                    $group:
                        {       _id:   {cliente: $cliente,
                                        condPago:"$condPago"} ,
                                total: { $sum: "$totalFactura" }
                        }
                    }
                    ])
```

`Retorna el monto total de todas las facturas de la colección agrupadas por el cliente y condición de pago`

## Ejemplos de uso en comparación con sentencias SQL

**SQL:**

```
SELECT cliente,  COUNT(*) AS total FROM facturas
GROUP BY cliente HAVING COUNT(*)>3
```

**Mongo:**

```
db.facturas.aggregate([
                    {
                    $group:
                        {       _id: $cliente,
                                total: { $sum: 1}
                        }
                    },{
                      $match:
                        {
                                total: {$gt:3}
                        }
                    }
                    ])
```

Retorna la cantidad total de facturas por el cliente, cuando mayor que 3

## Ejemplos de uso en comparación con sentencias SQL

**SQL:**

```
SELECT cliente, fechaEmision, SUM(totalFactura) AS total FROM facturas
GROUP BY cliente, fechaEmision HAVING SUM(totalFactura) > 10000
```

**Mongo:**

```
db.facturas.aggregate([
                    { $group:
                        { _id:
                            { cust_id: "$cliente",
                              ord_date: "$fechaEmision" },
                        total: { $sum: "$totalFactura" }
                         }
                    },
                    { $match:
                            { total: { $gt: 10000 } }
                     } ])
```

## Ejemplos de uso en comparación con sentencias SQL

**SQL:** SELECT cliente, SUM(totalFactura) AS total FROM facturas WHERE condPago='CONTADO' GROUP BY cliente

**Mongo:**

```
db.facturas.aggregate(
                    [
                    { $match: {  condPago: “CONTADO” } }
                    ,
                    { $group:
                            { _id:  "$cliente" ,
                            total: { $sum: "$totalFactura" }
                             }
                    }
                    ])
```

## Ejemplos de uso en comparación con sentencias SQL

**SQL:** SELECT cliente, SUM(totalFactura) AS total FROM facturas WHERE condPago='CONTADO' GROUP BY cliente HAVING SUM(totalFactura) > 50000

**Mongo:**

```
db.facturas.aggregate(
[
{ $match: {  condPago: “CONTADO” } }
,
{ $group:
        { _id:  "$cliente" ,
        total: { $sum: "$totalFactura" }
         }
},
{ $match: { total: { $gt: 50000 } } }
])
```

## Ejemplos de uso en comparación con sentencias SQL

**SQL:** ?????

**Mongo:**
```
db.facturas.aggregate([
                    { $project: {"item.cantidad":1, "cliente.region":1}},
                    { $unwind: "$item" },
                    { $group:
                                { _id: "$cliente",
                                cant: { $sum: "$item.cantidad" }
                                }
                     }
                    ])
```

## Ejemplos de uso en comparación con sentencias SQL

**SQL:** ?????

**Mongo:**

```
db.facturas.aggregate([
                    { $group:
                            { _id:
                                    { cust_id: "$cliente",
                                    fechaE: "$fechaEmision" }
                             }
                    },
                    { $group:
                            { _id: null,
                            count: { $sum: 1 }
                            }
                    }
                    ])
```

## ¿Que retorna la siguiente consulta sobre nuestra colección pizzas?

```
db.pizzas.aggregate( [
  {
    $match:
        {
      size:  "medium"
        }
  },
  {
    $group:
    {
      _id: {$toUpper:"$type"},
     total: { $sum: { $multiply: [ "$price", "$stats.sales" ] } },
      promedio: { $avg: "$stats.sales" }
    }
  },
  {
    $sort: { total: -1 }
  }
 ] )
```

## ¿Que retorna la siguiente consulta sobre nuestra colección pizzas?

```
db.pizzas.aggregate( [
  { $match:
        {
       size:  "medium"
        }
  },
  { $group:
    {
       _id:"$type",
       total: { $sum: { $multiply: [ "$price", "$stats.sales" ] } },
       promedio: { $avg: "$stats.sales" }
    }
  },
  { $sort: {total: -1 }
  },
  { $project : {_id:0,
                "Tipo": {$toUpper:"$_id"},
                "Total ventas": "$total",
                "Promedio ventas":"$promedio"
                }
   } ] )
```