# Pipeline de Agregación

Bases de Datos 2022

# Pipeline

## ¿Qué es un Pipeline?


## Pipeline en Unix

$command_1$ | $command_2$ | $command_3$ | ... | $command_N$

```
$ cat urls.txt
https://www.mongodb.com/docs/manual/tutorial/query-array-of-documents/
https://askubuntu.com/questions/1424501/unable-to-install-mongodb-in-ubuntu
https://www.mongodb.com/docs/manual/crud/
https://famaf.aulavirtual.unc.edu.ar/
https://dev.mysql.com/doc/refman/8.0/en/subqueries.html
https://dev.mysql.com/doc/refman/8.0/en/
https://dev.mysql.com/doc/
http://twitter.com/
$ cat urls.txt | sed -E "s/^https?:\/\/([^\/]+).*/\1/" | sort | uniq -c | sort -n -r
      3 dev.mysql.com
      2 www.mongodb.com
      1 twitter.com
      1 famaf.aulavirtual.unc.edu.ar
      1 askubuntu.com
```

# Pipeline de agregación

- Es una secuencia de una o más stages que procesan documentos


- Método Aggregate

db.<collection>.aggregate( pipeline, <options> )

db.<collection>.aggregate(

[ { <stage1> }, { <stage2> }, … , { <stageN> } ],

<options>

)

- Pipeline y stages de agregación

```
db.companies.aggregate( [
        { $match: { founded_year: 2004 } },
        { $sort: { name: 1} },
        { $limit: 5 },
        { $project: { _id: 0, name: 1 } }
] )
```

# Expresiones de agregación

{ <**operator**>: [ { argument$_1$ }, { argument$_2$ }, ... ] }

{ <**operator**>: { argument } }

- Expresiones booleanas
  - $and $not $or
- Expresiones de comparación
  - $cmp $eq $gt $gte $lt $lte $ne
- Expresiones aritméticas
  - $add $subtract $divide $abs ...
- Expresiones de arreglos
  - $arrayElemAt $first $last $size ...
  - $concatArray **$filter $map $reduce** ...
- Expresiones de conjuntos
  - $setDifference $setUnion $setIsSubset ...
- Expresiones condicionales
  - $cond $ifNull $switch
- Expresiones de fechas
  - $year $month $dateAdd $dateDiff ...
- Expresiones de strings
  - $concat $split $substr $dateFromString ...
- Expresiones de tipos
  - $convert $isNumber $type ...
- Field Path
  - "$<field>" (= "$$CURRENT.<field>")
- Mas expresiones

# Stages de agregación

- $match
  - Filtra los documentos que pasan a la siguiente etapa
- $project
  - Cambia la forma de cada documento
- $skip
  - Omite los primero N documentos
- $limit
  - Limita el resultado a los primeros N documentos
- $sort
  - Ordena el flujo de documentos por uno o más campos
- $count
  - Retorna la cantidad de documentos
- $addFields
  - Agrega nuevos campos a cada documento

# Stages de agregación - MATCH

{ $match: { <query filter> } }

{ $match: { $expr: < aggregation expression> } }

- Colección monthlyBudget

```
db.monthlyBudget.insertMany([
        { _id: 1, category: "food", budget: 400, spent: 450 },
        { _id: 2, category: "drinks", budget: 100, spent: 150 },
        { _id: 3, category: "clothes", budget: 100, spent: 50 },
        { _id: 4, category: "misc", budget: 500, spent: 300 },
        { _id: 5, category: "travel", budget: 200, spent: 650 }
]);
```

- Match con operadores de selección

```
db.monthlyBudget.aggregate( [
   { $match: { "budget": { $gt: 400, $lt: 600} } }
] )
```

- Match con expresiones de agregación

```
db.monthlyBudget.aggregate( [
   { $match: { $expr: { $gt: [ "$spent" , "$budget" ] } } }
] )
```

# Stages de agregación - PROJECT

{ **$project**: { <specifications> } }

{ **$project**: { <field$_1$>: < 1 or true>, <field$_2$>: < 0 or false>, <field$_3$>: < aggregation expression>, ... } }

- Colección monthlyBudget

```
db.monthlyBudget.insertMany([
        { _id: 1, category: "food", budget: 400, spent: 450 },
        { _id: 2, category: "drinks", budget: 100, spent: 150 },
        { _id: 3, category: "clothes", budget: 100, spent: 50 },
        { _id: 4, category: "misc", budget: 500, spent: 300 },
        { _id: 5, category: "travel", budget: 200, spent: 650 }
]);
```

- Project con expresiones de agregación

```
db.monthlyBudget.aggregate( [
   { $match: { $expr: { $gt: [ "$spent" , "$budget" ] } } },
   {
     $project: {
        cat_prefix: { $substr: [ "$category", 0, 3 ] },
        excess: { $subtract: [ "$spent" , "$budget" ] },
        _id: 0
     }
   }
] )
```

# Stages de agregación - SORT SKIP LIMIT COUNT

- Métodos del cursor: SORT, SKIP y LIMIT

```
db.monthlyBudget.find(
   { $expr: { $gt: [ "$spent" , "$budget" ] } },
   {
      cat_prefix: { $substr: [ "$category", 0, 3 ] },
      excess: { $subtract: [ "$spent" , "$budget" ] },
      _id: 0
   }
).sort(
   { excess: -1, excess: 1}
).skip( 0
).limit(1)
```

- Métodos del cursor: COUNT

```
db.monthlyBudget.find( { } ).count()
```

- Stages: SORT, SKIP y LIMIT

```
db.monthlyBudget.aggregate( [
   { $match: { $expr: { $gt: [ "$spent" , "$budget" ] } } },
   {
     $project: { cat_prefix: { $substr: [ "$category", 0, 3 ] },
         excess: { $subtract: [ "$spent" , "$budget" ] }, _id: 0 }
   },
   { $sort: { excess: -1, cat_prefix: 1 } },
   { $skip: 0 },
   { $limit: 1 }
] )
```

- Stages: COUNT

```
db.monthlyBudget.aggregate( [ { $count: "numOfItems" } ] )
```

# Stages de agregación - ADDFIELDS

{ **$addFields**: { <newField$_1$>: < aggregation expression>, ... } }

## ➢ Colección survey

```
db.survey.insertMany([
    { _id: 1, results: [ { product: "abc", score: 10 }, { product: "xyz", score: 5 } ] },
    { _id: 2, results: [ { product: "abc", score: 9 }, { product: "xyz", score: 8 } ] },
    { _id: 3, results: [ { product: "abc", score: 6 }, { product: "xyz", score: 3 } ] }
])
```

## ➢ Ejemplo de addFields

```
db.survey.aggregate( [
   {
      $addFields: { "avg_score": { $avg: "$results.score" } }
   }, {
      $match: { avg_score: { $gte: 7 } }
   }, {
      $project: {"avg_score": 1}
   }
] )
```

# IT’S DEMO TIME

# Stages de agregación (2)

- $unwind
  - Deconstruye un campo arreglo en el documento y crea documentos separados para cada elemento en el arreglo
- $replaceRoot
  - Reemplaza el documento por un documento anidado especificado
- $group
  - Agrupa los documentos por una expresión especificada y aplica las expresiones acumuladoras
- $unionWith
  - Realiza la unión de dos colecciones
- $out
  - Almacena el resultado del pipeline en una colección
- $lookup
  - Realiza un left join a otra colección

# Stages de agregación (2) - UNWIND

{ **$unwind**: <field path> }

## Colección survey

```
db.survey.insertMany([
    { _id: 1, results: [ { product: "abc", score: 10 }, { product: "xyz", score: 5 } ] }
])
```

## Ejemplo de unwind

```
db.survey.aggregate( [
   {
      $unwind: "$results"
   }
] )
```

## Resultado del pipeline

```
[
 { _id: 1, results: { product: 'abc', score: 10 } },
 { _id: 1, results: { product: 'xyz', score: 5 } }
]
```

# Stages de agregación (2) - ReplaceRoot

{ **$replaceRoot**: { newRoot: < replacementDocument > } }

## ➢ Colección survey

```
db.survey.insertMany([
    { _id: 1, results: [ { product: "abc", score: 10 }, { product: "xyz", score: 5 } ] },
    { _id: 2, results: [ { product: "abc", score: 9 }, { product: "xyz", score: 8 } ] },
    { _id: 3, results: [ { product: "abc", score: 6 }, { product: "xyz", score: 3 } ] }
])
```

## ➢ Ejemplo de replaceRoot

```
db.survey.aggregate( [
   { $unwind: "$results"  },
   {
      $match: { "results.score": { $gte: 9 } }
   },
   {
      $replaceRoot: { newRoot: "$results" }
   }
] )
```

# Stages de agregación (2) - GROUP

```
{
    $group: {
        _id: <expression>,                          // single or group key
        <field1>: { <accumulator1>: <expression1>,  // accumulator expression: $avg, $max, $min, $stdDevPop,
        ...                                         // $count, $sum, $first, $addToSet, $push, ....
    }
}
```

➢ Colección sales

```
db.sales.insertMany([
   { _id: 1, item: "abc", price: 10, qty: 2, date: ISODate("2014-03-01T08:00:00Z") },
   { _id: 2, item: "jkl", price: 20, qty: 1, date: ISODate("2014-03-01T09:00:00Z") },
   { _id: 3, item: "xyz", price: 5, qty: 10, date: ISODate("2014-03-15T09:00:00Z") },
   { _id: 4, item: "xyz", price: 5, qty: 20, date: ISODate("2014-04-04T11:21:39Z") },
   { _id: 5, item: "abc", price: 10, qty: 10, date: ISODate("2014-04-04T21:23:13Z") },
   { _id: 6, item: "def", price: 7.5, qty: 5, date: ISODate("2015-06-04T05:08:13Z") },
   { _id: 7, item: "def", price: 7.5, qty: 10, date: ISODate("2015-09-10T08:43:00Z") },
   { _id: 8, item: "abc", price: 10, qty: 5, date: ISODate("2016-02-06T20:20:13Z") }
])
```

➢ Ejemplo de group donde _id es null y expresiones de acumulador

```
db.sales.aggregate([
   {
      $group: {
         _id: null,
         totalQuanty: { $sum: "$qty"},
         count: { $count: {}}
      }
   }
])
```

➢ Resultado del pipeline

```
[ { _id: null, totalQuanty: 63, count: 8 } ]
```

# Stages de agregación (2) - GROUP

- Agrupar por un solo campo

```
db.sales.aggregate( [
   {
      $group: {
         _id: "$item",
         tamount: { $sum: { $multiply: [ "$price", "$qty" ] } },
      }
   }
] )
```

- Resultado del pipeline

```
[
  { _id: 'def', amount: 112.5 },
  { _id: 'jkl', amount: 20 },
  { _id: 'abc', amount: 170 },
  { _id: 'xyz', amount: 150 }
]
```

- Agrupar por varios campos

```
db.sales.aggregate( [
   {
      $group: {
         _id: { "year": {$year: "$date"}, month: {$month: "$date" } },
         totalQuantity: { $sum: "$qty" },
         count: { $sum: 1 }
      }
   }
] )
```

- Resultado del pipeline

```
[
  { _id: { year: 2014, month: 3 }, totalQuantity: 13, count: 3 },
  { _id: { year: 2015, month: 9 }, totalQuantity: 10, count: 1 },
  { _id: { year: 2015, month: 6 }, totalQuantity: 5, count: 1 },
  { _id: { year: 2016, month: 2 }, totalQuantity: 5, count: 1 },
  { _id: { year: 2014, month: 4 }, totalQuantity: 30, count: 2 }
]
```

# Stages de agregación (2) - UnionWith

{ **$unionWith**: "<collection>" }

{ **$unionWith**: { coll: "<collection>", pipeline: [ { <stage$_1$> }, ... ] } }

## Colecciones cats y dogs

```
db.cats.insertMany([
        { _id: 1, name: "Fluffy", type: "Cat", weight: 5 },
        { _id: 2, name: "Scratch", type: "Cat", weight: 3 },
        { _id: 3, name: "Meow", type: "Cat", weight: 7 }
] )

db.dogs.insertMany([
        { _id: 1, name: "Wag", type: "Dog", weight: 20 },
        { _id: 2, name: "Bark", type: "Dog", weight: 10 },
        { _id: 3, name: "Fluffy", type: "Dog", weight: 40 }
] )
```

## Ejemplo de unionWith

```
db.cats.aggregate( [
   {
      $unionWith: {
         coll: "dogs",
         pipeline: [ { $match: { weight: { $lt: 30 } } } ]
      }
   }
] )
```

## Resultado del pipeline

```
[
  { _id: 1, name: 'Fluffy', type: 'Cat', weight: 5 },
  { _id: 2, name: 'Scratch', type: 'Cat', weight: 3 },
  { _id: 3, name: 'Meow', type: 'Cat', weight: 7 },
  { _id: 1, name: 'Wag', type: 'Dog', weight: 20 },
  { _id: 2, name: 'Bark', type: 'Dog', weight: 10 }
]
```

# Stages de agregación (2) - OUT

{ **$out**: { db: "<outDatabase>", coll: "<outCollection>"} }

- Colecciones cats y dogs

```
db.cats.insertMany([
        { _id: 1, name: "Fluffy", type: "Cat", weight: 5 },
        { _id: 2, name: "Scratch", type: "Cat", weight: 3 },
        { _id: 3, name: "Meow", type: "Cat", weight: 7 }
] )

db.dogs.insertMany([
        { _id: 1, name: "Wag", type: "Dog", weight: 20 },
        { _id: 2, name: "Bark", type: "Dog", weight: 10 },
        { _id: 3, name: "Fluffy", type: "Dog", weight: 40 }
] )
```

- Ejemplo de replaceRoot

```
db.cats.aggregate( [
   {
      $unionWith: {
         coll: "dogs",
         pipeline: [ { $match: { weight: { $lt: 30 } } } ]
      }
   },
   {  $unset: "_id"   },
   {
      $out: { db: "samples", coll: "pets" }
   }
] )

db.pets.find()
```

# Stages de agregación (2) - LOOKUP - Condición de JOIN simple

```
{
    $lookup: {
        from: <collection to join>,
        localField: <field from the input documents>
        foreignField: <field from the documents of the "from" collection>,
        as : <output array field>
    }
}
```

- Colección posts

```
db.posts.insertMany( [
    { _id: 1, "author": "Jim", "likes": 5 },
    { _id: 2, "author": "Jim", "likes" : 2 },
    { _id: 3, "author": "Joe", "likes": 3 }
] )
```

- Colección posts

```
db.comments.insertMany( [
    { "comment": "great read", "likes": 3, post_id: 1 },
    { "comment": "good info", "likes": 0, post_id: 2 },
    { "comment": "i liked this post", "likes": 12, post_id: 2 },
    { "comment": "not my favorite", "likes": 8, post_id: 3 },
    { "comment": null, "likes": 0, post_id: 4 }
] )
```

# Stages de agregación (2) - LOOKUP - Condición de JOIN simple

- Lookup con condición de join simple

```
db.posts.aggregate( [
    {
        $lookup: {
            from: "comments",
            localField: "_id",
            foreignField: "post_id",
            as: "cmts"
        }
    }
] )
```

- Resultado del pipeline

```
[
 {
  _id: 1, author: 'Jim', likes: 5,
  cmts: [ { _id: ObjectId, comment: 'great read', likes: 3, post_id: 1 } ]
 },
 {
  _id: 2, author: 'Jim', likes: 2,
  cmts: [
   { _id: ObjectId, comment: 'good info', likes: 0, post_id: 2 },
   { _id: ObjectId, comment: 'i liked this post', likes: 12, post_id: 2 }
  ]
 },
 {
  _id: 3, author: 'Joe', likes: 3,
  cmts: [ { _id: ObjectId, comment: 'not my favorite', likes: 8, post_id: 3 } ]
 }
]
```

# Stages de agregación (2) - LOOKUP - Multiple condiciones

```
{
        $lookup: {
                from: <joined collection>,
                let: { <var_1>: <expression>, … , <var_N>: <expression> },
                pipeline: [ <pipeline to run joined collection> ],
                as : <output array field>
        }
}
```

# Stages de agregación (2) - LOOKUP - Multiple condiciones

- Lookup con condición de join múltiple y subquery

```
db.posts.aggregate( [
   {
      $lookup: {
         from: "comments",
         let: { post_likes: "$likes", post_id: "$_id" },
         pipeline: [
            {
               $match: {
                  $expr: {
                     $and: [
                        { $eq: ["$post_id", "$$post_id" ] },
                        { $gt: [ "$likes", "$$post_likes" ] }
                     ]
                  }
               }
            }
         ],
         as: "cmts"
      }
   }
] )
```

- Resultado del pipeline

```
[
   { _id: 1, author: 'Jim', likes: 5, cmts: [] },
   {
      _id: 2, author: 'Jim', likes: 2,
      cmts: [
         {
            _id: ObjectId("635b467558c2ead4c68a4880"),
            comment: 'i liked this post',
            likes: 12,
            post_id: 2
         }
      ]
   },
   {
      _id: 3, author: 'Joe', likes: 3,
      cmts: [
         {
            _id: ObjectId("635b467558c2ead4c68a4881"),
            comment: 'not my favorite',
            likes: 8,
            post_id: 3
         }
      ]
   }
]
```

# Vistas

db.createView( "<viewName>", "<source>", [ <pipeline> ] )

## Colecciones students

```
db.students.insertMany( [
   { sID: 22001, name: "Alex", year: 1, score: 4.0 },
   { sID: 21001, name: "bernie", year: 2, score: 3.7 },
   { sID: 20010, name: "Chris", year: 3, score: 2.5 },
   { sID: 22021, name: "Drew", year: 1, score: 3.2 },
   { sID: 17301, name: "harley", year: 6, score: 3.1 },
   { sID: 21022, name: "Farmer", year: 1, score: 2.2 },
   { sID: 20020, name: "george", year: 3, score: 2.8 },
   { sID: 18020, name: "Harley", year: 5, score: 2.8 },
] )
```

## Ejemplo de createView

```
db.createView(
      "firstYears",
      "students",
      [ { $match: { year: 1 } } ]
)
```

```
db.firstYears.find( { }, { _id: 0 } )
```

```
[
  { sID: 22001, name: 'Alex', year: 1, score: 4 },
  { sID: 22021, name: 'Drew', year: 1, score: 3.2 },
  { sID: 21022, name: 'Farmer', year: 1, score: 2.2 }
]
```

# Temas a estudiar

- Próxima clase
  - Modelado de datos
  - Validación de esquemas
- Referencias
  - Pipeline de agregación
  - Vistas