MongoDB Enterprise 4.0.10

CURSO FUNDAMENTOS DE MONGODB

Instructor: Carlos Carreño
Email: ccarrenovi@Gmail.com

CRUD en MongoDB

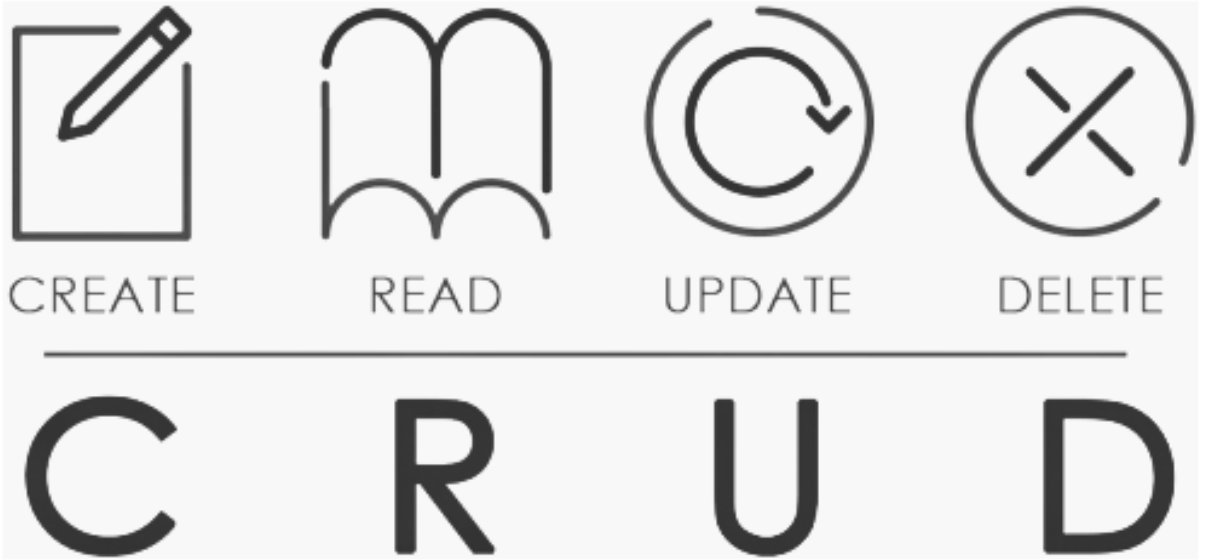
- ¿Qué es CRUD?
- Create
  - Insert()
  - InsertMany()
  - save()
- Read
  - Find()
  - findOne()
  - Operadores de comparación: $gt, $lt
  - Operadores de búsqueda: $in, $match, $elementMatch
  - Operadores Lógicos: $or, $and
  - explain() Comprensión de un plan de ejecución de Consultas
  - hint()
  - Consultas con Aggregate y Map-Reduce
- Operadores de agregación: $match, $group, $sum, $project, $switch, $lookup,

… continue

- $unwind Update
  - Update ()
  - updateOne()
  - findOneAndUpdate()
  - Drop()
  - Arrays de Objetos
  - Funciones que manejan Arrays: slice, push, pull, arrayFilter gridFS
  - Almacenamiento de archivos binarios en Colecciones Export
  - mongoexport (incluir parámetros de filtro) Import
  - mongoimport

¿Qué es CRUD?

- C – create (crear)
- R – read (leer)
- U – update (actualizar)
- D – delete (eliminar)



Insertar documentos

- Para insertar documentos utiliza el comando db.<nombre de la colección>.insert({..}). Si la colección no existe, mongoDB la crea.

```
> db.customer.insert({name:'Kevin P.',lastname:'Ryan',phone:'+1 345678',age:40,address:'20 Renfer, Malibu'});
WriteResult({ "nInserted" : 1 })
> db.customer.insert({name:'Eliot',lastname:'Horowitz',phone:'+1 445673',age:42,address:'34 Bronze, NY'});
WriteResult({ "nInserted" : 1 })
```

Insertar varios documentos

- Utiliza la función insertMany

```
db.products.insertMany( [
      { _id: 10, item: "large box", qty: 20 },
      { _id: 11, item: "small box", qty: 55 },
      { _id: 12, item: "medium box", qty: 30 }
   ] );
```

Consultando datos de la colección

- Utiliza la función find(…)

```
use devmongodb
switched to db devmongodb
show collections
customer
 db.customer.find();
{ "_id" : ObjectId("5daa4d1f56aead7367e49658"), "name" : "Kevin P.", "lastname" : "Ryan", "phone" : "+1 345678", "age" : 40, "address" : "20 Renfer, Malibu" }
{ "_id" : ObjectId("5daa4d7656aead7367e49659"), "name" : "Eliot", "lastname" : "Horowitz", "phone" : "+1 445673", "age" : 42, "address" : "34 Bronze, NY" }
{ "_id" : ObjectId("5daa4db656aead7367e4965a"), "name" : "Dwight", "lastname" : "Merriman", "phone" : "+1 345671", "age" : 40, "address" : "19 Avenued, NY" }
```

Consultando datos con parámetros

- Utiliza la función find con uno o mas argumentos

```
db.customer.find({age:40});
```

{ "_id" : ObjectId("5daa4d1f56aead7367e49658"), "name" : "Kevin P.", "lastname" : "Ryan", "phone" : "+1 345678", "age" : 40, "address" : "20 Renfer, Malibu" }

{ "_id" : ObjectId("5daa4db656aead7367e4965a"), "name" : "Dwight", "lastname" : "Merriman", "phone" : "+1 345671", "age" : 40, "address" : "19 Avenued, NY" }

```
db.customer.find({age:{$gt:40}});
```

{ "_id" : ObjectId("5daa4d7656aead7367e49659"), "name" : "Eliot", "lastname" : "Horowitz", "phone" : "+1 445673", "age" : 42, "address" : "34 Bronze, NY" }

Operadores de comparación

- Operadores de comparación para diferentes tipos de datos en BSON

Name	Description
`$eq`	Matches values that are equal to a specified value.
`$gt`	Matches values that are greater than a specified value.
`$gte`	Matches values that are greater than or equal to a specified value.
`$in`	Matches any of the values specified in an array.
`$lt`	Matches values that are less than a specified value.
`$lte`	Matches values that are less than or equal to a specified value.
`$ne`	Matches all values that are not equal to a specified value.
`$nin`	Matches none of the values specified in an array.

Eliminar documentos

- Eliminar un documento dado un _id

```
db.customer.remove({_id:ObjectId("5daa4d7656aead7367e49659")});
WriteResult({ "nRemoved" : 1 })
db.customer.find();
{ "_id" : ObjectId("5daa4d1f56aead7367e49658"), "name" : "Kevin P.", "lastname" : "Ryan", "phone" : "+1 345678", "age" : 40, "address" : "20 Renfer, Malibu" }
{ "_id" : ObjectId("5daa4db656aead7367e4965a"), "name" : "Dwight", "lastname" : "Merriman", "phone" : "+1 345671", "age" : 40, "address" : "19 Avenued, NY" }
```

Eliminar documentos

- Eliminar documentos mediante una comparación

```
db.customer.remove({age:40});
WriteResult({ "nRemoved" : 2 })
db.customer.find();
{ "_id" : ObjectId("5daa538356aead7367e4965b"), "name" : "Eliot", "lastname" : "Horowitz", "phone" : "+1 445673", "age" : 42, "address" : "34 Bronze, NY" }
```

Editar documentos

- Para editar documentos usa la función update

```
db.customer.update({age:42},{phone:'+1 345678'});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
db.customer.find();
{ "_id" : ObjectId("5daa538356aead7367e4965b"), "phone" : "+1 345678" }
{ "_id" : ObjectId("5daa53bb56aead7367e4965c"), "name" : "Kevin P.", "lastname" : "Ryan", "phone" : "+1 345678", "age" : 40, "address" : "20 Renfer, Malibu" }
{ "_id" : ObjectId("5daa53bf56aead7367e4965d"), "name" : "Dwight", "lastname" : "Merriman", "phone" : "+1 345671", "age" : 40, "address" : "19 Avenued, NY" }
```

Editar documentos y agregar campos

- Para editar documentos y agregar campos utiliza update y $set

```
db.customer.update({age:40},{$set:{country:'USA',hobby:'Tennis'}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
db.customer.find();
{ "_id" : ObjectId("5daa53bb56aead7367e4965c"), "name" : "Kevin P.", "lastname" : "Ryan", "phone" : "+1 345678", "age" : 40, "address" : "20 Renfer, Malibu", "country" : "USA", "hobby" : "Tennis" }
{ "_id" : ObjectId("5daa53bf56aead7367e4965d"), "name" : "Dwight", "lastname" : "Merriman", "phone" : "+1 345671", "age" : 40, "address" : "19 Avenued, NY" }
{ "_id" : ObjectId("5daa583256aead7367e49661"), "name" : "Eliot", "lastname" : "Horowitz", "phone" : "+1 445673", "age" : 42, "address" : "34 Bronze, NY" }
```

Nota: Observa que el comando solo actualiza la primera coincidencia.

Actualizando varios documentos

- Utiliza la función updateMany

```
{ "_id" : 1, "name" : "Central Perk Cafe", "places" : 3 }
{ "_id" : 2, "name" : "Rock A Feller Bar and Grill", "places" : 2 }
{ "_id" : 3, "name" : "Empire State Sub", "places" : 5 }
{ "_id" : 4, "name" : "Pizza Rat's Pizzaria", "places" : 8 }
```

```
db.restaurant.updateMany(
    { places: { $gt: 4 } },
    { $set: { "Review" : true } }
  );
```

Actualización masiva de documentos

- Se puede utilizar update para la actualización masiva, para ello la propiedad multi debe tener el valor de true

```
db.products.update({stock:20},
        {
          $set:{stock:40}
         },
         {multi:true}
);
```

Función pretty()

- Por defecto mongoDB, muestra los datos como una lista, para mejorar el formato de salida puedes usar la función pretty()

```
db.customer.find().pretty();
{
        "_id" : ObjectId("5daa53bb56aead7367e4965c"),
        "name" : "Kevin P.",
        "lastname" : "Ryan",
        "phone" : "+1 345678",
        "age" : 40,
        "address" : "20 Renfer, Malibu",
        "country" : "USA",
        "hobby" : "Tennis"
}
```

Documentos embebidos con arrays

- Un documento puede contener varios documentos embebidos, para este propósito se utiliza un array en BSON se representa por los corchetes []. Ejemplo: En los siguientes documentos, el documento **comments** esta embebido en los documentos de **products**

Products documents:

{name:'hard disc SDD 480 GB',stock:20,price:128.50, **comments**:[{uid:1,text:"buen producto"},{uid:2,text:"llego incompleto"}]},
{name:'processor core i7 8th 2.3 GHZ',stock:10,price:1250.0, **comments**:[{uid:1,text:"relacion precio producto bueno"}]},
{name:'Video card GTI 1050 4GB',stock:15,price:800.50},
{name:'Monitor LG HDMI full hd',stock:10,price:1300.0, **comments**:[{uid:1,text:"buena resolucion"}]}

Agregando un documento embebido

- Para agregar un nuevo documento embebido utiliza la operación $push

```
db.products.update(
 { "_id" : ObjectId("5dab225a81e628f3a93ce363")},
 {
  $push:{
    comments:{uid:4,text:"excelente producto"}
   }
 }
);
```

Incrementar datos con $inc

- Se puede incrementar un campo numérico, para ello utiliza la operación $inc

```
db.products.update(
  { "_id" : ObjectId("5dab225a81e628f3a93ce363")},
  {
    $inc:{
       stock:+10
     }
  }
);
```

Ordenar resultados de la consulta

- Utiliza la función sort

```
db.products.find().sort({stock:-1}).pretty();
```

1 ordena de manera ***ascendente***

-1 ordena de manera ***descendente***

Consultas con filtros tipo LIKE

- Para los patrones de las expresiones regulares MongoDB utiliza "*Perl Compatible Regular Expressions*" (PCRE). De esta forma tendremos las siguientes similitudes con los patrones LIKE.

cadena%	/^cadena/	Que empiecen
%cadena%	/cadena/	Que contengan
%cadena	/cadena$/	Que terminen

```
db.products.find({"comments.text":/buen/}).pretty();
```

Consultas tipo Between

- Para realizar tipo de consultas between usamos los operadores de comparación

```
db.products.find({stock:{$gt:15,$lt:30}}).pretty();
```

Recordar:

$gt	>	mayor que
$lt	<	menor que
$gte	>=	mayor igual que
$lte	<=	menor igual que
$eq	==	igual que
$ne	!=	diferente que

Ocultar un campo en el resultado

- Puedes ocultar un campo en el resultado con el valor 0
- Ejemplo: La siguiente operación de búsqueda oculta el campo comments.

```
db.products.find({stock:{$gt:15,$lt:30}},{comments:0}).pretty();
```

Contando los resultados

- Para contar los documentos del resultado utiliza la función count()

```
db.products.find({stock:{$eq:20}}).count();
```

Creando un documento embebido

- Utiliza la función update y la operación $set

- Ejemplo: Se crea el documento ***provider*** en el documento ***products***

```
db.products.update({"_id" : ObjectId("5dab225a81e628f3a93ce362")},
    {
     $set:{provider:{country:"China",phone:"+86 3456789"}}
    }
);
```

Agregando una propiedad al documento embebido

- Utiliza la función update y el operador $set

```
db.products.update({"provider.country":'China'},
        {
            $set:{"provider.email":'leeyuang@aliexpress.com'}
        }
);
```

Eliminando una propiedad al documento embebido

- Utiliza la función update y el operador $unset

```
db.products.update({"provider.country":'China'},
        {
         $unset:{"provider.email":''}
        }
);
```

Eliminando una propiedad de un documento embebido en un array

- Ejemplo 1: Agregando la propiedad

```
db.products.update({"stock":10},
                   {$push:{ comments:{uid:3,text:"llego bien",tag:"normal"} }
});
```

- Ejemplo 2: Eliminando la propiedad

```
db.products.update({"stock":10},
                   {$pull:{ comments:{tag:"normal"} } },
{multi:true});
```

Eliminando una propiedad de un documento embebido en un array con el operador $in

- Ejemplo 3: Elimina un documento embebido del array que contenga determinada palabra o lista de palabras

```
db.products.update({"stock":40},
                   {$pull:{ comments:{text:{$in:[/excelente/]}} } },
{multi:true});

db.products.update({"stock":40},
                   {$pull:{ comments:{text:{$in:[/excelente/,/resolucion/]}} } },
{multi:true});
```

Eliminando la colección

- Utiliza la función drop()

```
db.customer.drop()
```

Borrar la base de datos

- Para eliminar una base de datos usa la función dropDatabase()

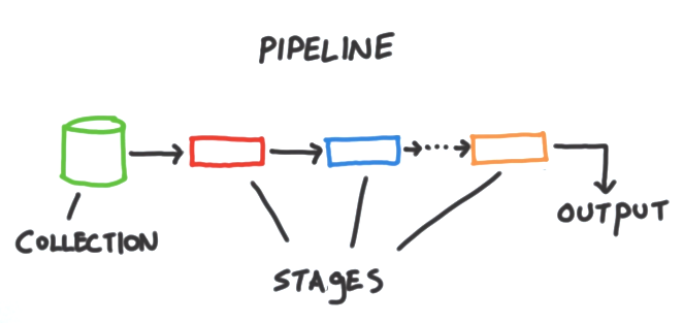
```
use devmongodb;
switched to db devmongodb
db.dropDatabase();
{ "dropped" : "devmongodb", "ok" : 1 }
```

Practica

- Practica 2 Base de Datos y Documentos en MongoDB

Aggregation Framework

- Es habitual tener que realizar consultas para agrupar datos y calcular valores a partir de ellos.
- En las bases de datos relacionales para agrupar usamos operadores como GROUP BY y para cálculos usamos AVG, SUM, COUNT, En MongoDB tenemos dos opciones: MapReduce y Aggregation Framework



Partes de una consulta de agregación

- Una consulta de agregación con Aggregation Framework tiene el siguiente formato

db.<collection>.aggregate([<pipeline>])

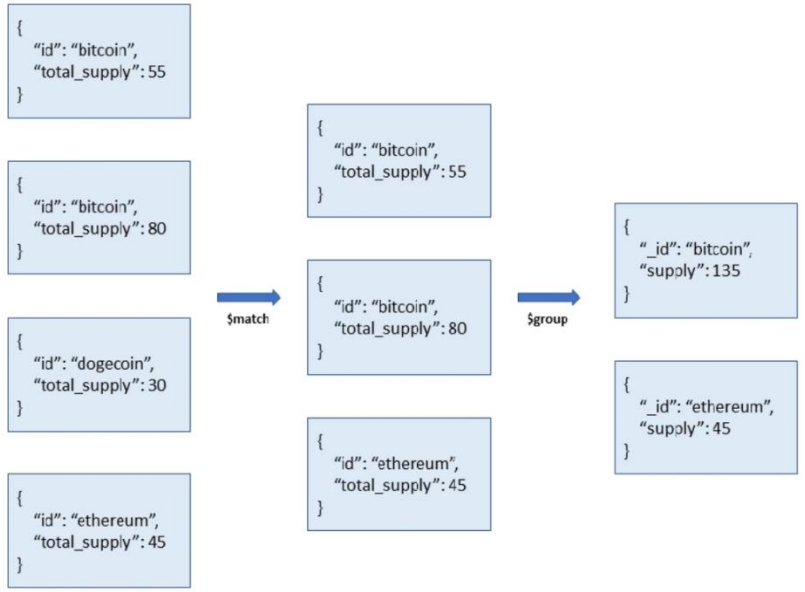
- Los ***pipelines*** o ***tuberías***, son similares a las que se utilizan en la línea de comandos de los sistemas Unix, pasando los resultados de un comando a otro para producir resultados de forma conjunta.

Pipelines de agregación

- ***$project*** : se utiliza para modificar el conjunto de datos de entrada, añadiendo, eliminando o recalculando campos para que la salida sea diferente.
- ***$match***: filtra la entrada para reducir el número de documentos, dejando solo los que cumplan las condiciones establecidas.
- ***$limit***: restringe el número de resultados al número indicado.
- ***$skip***: ignora un número determinado de registros, devolviendo los siguientes.
- ***$unwind***: convierte un array para devolverlo separado en documentos.
- ***$group***: agrupa documentos según una determinada condición.
- ***$sort***: ordena un conjunto de documentos según el campo especificado.
- ***$geoNear***: utilizado con datos geoespaciales, devuelve los documentos ordenados por proximidad según un punto geoespacial.

Pipelines

- Los pipelines pueden trabajar en conjunto




```
db.tickers.aggregate([
        {$match: {total_supply: {$gt: 30} } },
        {$group: {_id: "$id", supply: {$sum: "$total_supply"}}}
])
```

Operadores básicos de agregación

- Para realizar cálculos sobre los datos producidos por los *pipelines*, utilizamos las ***expresiones***. Las *expresiones* son funciones que realizan una determinada operación sobre un grupo de documentos, un array o un campo en concreto.
- ***$first***: Devuelve el primer valor de un campo en un grupo. Si el grupo no está ordenado, el valor mostrado será impredecible.
- ***$last***. Devuelve el último valor de un campo en un grupo. Si el grupo no está ordenado, el valor mostrado será impredecible.
- ***$max***. Devuelve el valor más alto de un determinado campo dentro un grupo.
- ***$min***. Devuelve el valor más pequeño de un determinado campo dentro de un grupo.
- ***$avg***. Calcula la media aritmética de los valores dentro del campo especificado y los devuelve.
- ***$sum***. Suma todos los valores de un campo y los devuelve.

Calculo en una agrupación

- Examinemos la siguiente muestra de datos:

```
{idcliente:"102",mount:456.0, dateorder:"2019-01-25", status:"delivered"},
{idcliente:"103",mount:234.0, dateorder:"2019-01-27", status:"complete"},
{idcliente:"101",mount:13456.0, dateorder:"2019-01-19", status:"pending"},
{idcliente:"103",mount:10456.0, dateorder:"2019-02-18", status:"delivered"},
{idcliente:"104",mount:21366.0, dateorder:"2019-03-21", status:"delivered"},
{idcliente:"105",mount:3457.0, dateorder:"2019-03-25", status:"complete"},
{idcliente:"101",mount:2016.0, dateorder:"2019-04-22", status:"pending"}
```

- Si fuera relacional y necesitáramos calcular la suma de los montos de las ordenes por cliente la consulta seria:

```
SELECT SUM(MOUNT) total  FROM ORDERS GROUP BY IDCLIENTE
```

- En MongoDB con agregaciones es:

```
db.orders.aggregate(
 [{
   $group:{_id:"$idcliente", total:{ $sum:"$mount" } }
}]);
```

Usando operadores en agregaciones

```
db.people.aggregate({
   $sort: {   name: 1    }
  }, {   $group: {
       _id: {
          age: "$age",
          gender: "$gender"
       },
       count: {
          $sum: 1
       },
       avgweight: {
          $avg: "$weight"
       }
   } });
```

Pipeline $project

- Este *pipeline* tiene una funcionalidad parecida a la que tiene el clásico *SELECT* en una consulta SQL. Con él, podremos cambiar los campos originales que tiene un documento añadiendo otros nuevos, eliminando, cambiando el nombre o añadiendo datos calculados.
- Ejemplo:

```
db.people.aggregate({
    $project: {
        isActive: 1,
        company: 1
    }
});
```

Pipeline $match

- El *pipeline $match* se utiliza para filtrar los documentos que se pasarán al siguiente pipeline de la consulta con **Aggregation Framework**

```
db.people.aggregate({
    $match: {
        isActive: true
    }
});
```

Pipeline $group

- Permite agrupar los documentos. Es importante destacar que *$group* siempre tiene que tener un campo *_id*. Es el campo por el que vamos a agrupar los resultados.

```
db.people.aggregate({
   $group: {
      _id: {
         gender: "$gender"
      },
      averageAge: {
         $avg: "$age"
      },
      count: {
         $sum: 1
      }
   }
});
```

Pipeline $sort

- En cualquier consulta de agregación que se precie, es probable que nos veamos en la necesidad de ordenar los resultados. Y es aquí dónde entra el pipeline *$sort*.

```
db.people.aggregate({
    $sort: {
        age: 1
    }
});
```

Prioridad en $sort

- Cuando ordenamos por campos de distintos tipos $sort sigue la siguiente prioridad:

1. Null
2. Valores numéricos (int, log, double)
3. Cadenas de caracteres
4. Objetos
5. Arrays
6. Datos binarios
7. ObjectID (como el campo _id que MongoDB genera para cada documento)
8. Boolean
9. Fechas
10. Expresiones regulares

Pipelines $limit y $skip

- Estos dos pipelines son muy sencillos de utilizar. Con *$limit*, reduciremos el número de documentos devueltos por la consulta hasta el número que indiquemos. Con *$skip* lo que haremos será ignorar un número de elementos determinado que no serán devueltos en la consulta.

```
db.people.aggregate({
    $limit: 3
});

db.people.aggregate({
    $skip: 2
});
```

$unwind

- Utilizando *$unwind* conseguiremos separar los elementos de un array, creando como resultado tantos documentos iguales como elementos tenga el array, pero incluyendo sólo el valor del array.

```
db.people.aggregate({
   $match: {
      name: "Yenny"
    } },
    {   $project: {
           name: 1,
           address: 1
     } },{
         $unwind: "$address"
    });
```

Operadores de agregación y comparación

- Este tipo de operadores de expresión se utilizan para comparar valores y devolver un resultado. Los operadores disponibles son los siguientes:
- $cmp: compara dos valores y devuelve un número entero como resultado. Devuelve -1 si el primer valor es menor que el segundo, 0 si son iguales y 1 si el primer valor es mayor que el segundo.
- $eq: compara dos valores y devuelve true si son equivalentes.
- $gt: compara dos valores y devuelve true si el primero es más grande que el segundo.
- $gte: compara dos valores y devuelve true si el primero es igual o más grande que el segundo.
- $lt: compara dos valores y devuelve true si el primero es menor que el segundo.
- $lte: compara dos valores y devuelve true si el primero es igual o menor que el segundo.
- $ne: compara dos valores y devuelve true si los valores no son equivalentes.

Operadores de agregación booleanos

Un operador booleano recibe parámetros booleanos y devuelve otro booleano como resultado. Los operadores booleanos que podemos utilizar son:

- *$and*: devuelve true si todos los parámetros de entrada son true.
- *$or*: devuelve true si alguno de los parámetros de entrada es true.
- *$not*: devuelve el valor contrario al parámetro de entrada. Si el parámetro es true, devuelve false. Si el parámetro es false, devuelve true.

Operadores de agregación condicionales

Los operadores condicionales en **MongoDB**se utilizan para verificar una expresión y según el valor de esta expresión, devolver un resultado u otro. En **Aggregation Framework**existen dos operadores condicionales:

- *$cond*: operador ternario que recibe tres expresiones. Si la primera es verdadera, devuelve la segunda. Si es falsa, devuelve la tercera. Muy similar, por ejemplo, al operador ternario de JavaScript
- *$ifNull*: operador que recibe dos parámetros y en el caso de que el primero sea null, devuelve el segundo. Muy similar al *IsNull* de SQL Server o al *NVL* de Oracle.

Operadores de agregación de cadenas

Los operadores de cadena son aquellos que se utilizan para realizar operaciones con strings. Son los siguientes:

- *$concat*: concatena dos o más strings.
- *$strcasecmp:* compara dos strings y devuelve un entero con el resultado. Devolverá un número positivo si el primer string es mayor, un número negativo si el segundo es mayor o un 0 en el caso de que sean iguales.
- *$substr:* crea un substring con una cantidad determinada de caracteres.
- *$toLower:* convierte el string a minúsculas.
- *$toUpper:* convierte el string a mayúsculas.

Operadores de agregación y fechas

Los operadores de fecha se utilizan para realizar operaciones con campos de tipo fecha. Tenemos disponibles los siguientes operadores:

- *$dayOfYear*: convierte una fecha a un número entre 1 y 366.
- *$dayOfMonth*: convierte una fecha a un número entre 1 y 31.
- *$dayOfWeek*: convierte una fecha a un número entre 1 y 7.
- *$year:* convierte una fecha a un número que representa el año (2000,1998 etc.)
- *$month*: convierte una fecha a un número entre el 1 y el 12.
- *$week*: convierte una fecha a un número entre 0 y 53.
- *$hour:* convierte una fecha a un número entre 0 y 23.
- *$minute*: convierte una fecha a un número entre 0 y 59
- *$second*: convierte una fecha a un número entre 0 y 59. Aunque puede ser 60 para contar intervalos.
- *$millisecond:* devuelve los milisegundos de la fecha, con un número entre 0 and 999.

Mongo Export

- Sintaxis

```
$ mongoexport
Export MongoDB data to CSV, TSV or JSON files.

options:
  -h [ --host ] arg         mongo host to connect to ( <set name>/s1,s2 for
  -u [ --username ] arg     username
  -p [ --password ] arg     password
  -d [ --db ] arg           database to use
  -c [ --collection ] arg   collection to use (some commands)
  -q [ --query ] arg        query filter, as a JSON string
  -o [ --out ] arg          output file; if not specified, stdout is used
```

```
$ mongoexport -d webmitta -c domain -o domain-bk.json
connected to: 127.0.0.1
exported 10951 records
```

Mongo Import

- Sintaxis

```
$ mongoimport
connected to: 127.0.0.1
no collection specified!
Import CSV, TSV or JSON data into MongoDB.

options:
  -h [ --host ] arg       mongo host to connect to ( <set name>/s1,s2 for sets)
  -u [ --username ] arg   username
  -p [ --password ] arg   password
  -d [ --db ] arg         database to use
  -c [ --collection ] arg collection to use (some commands)
  -f [ --fields ] arg     comma separated list of field names e.g. -f name,age
  --file arg              file to import from; if not specified stdin is used
  --drop                  drop collection first
  --upsert                insert or update objects that already exist
```

```
$ mongoimport -d webmitta2 -c domain2 --file domain-bk.json
connected to: 127.0.0.1
Wed Apr 10 13:26:12 imported 10903 objects
```

Practica

- Practica 3 Utilizando Aggregation Framework

Referencias

- https://platzi.com/contributions/introduccion-al-pipeline-de-agregacion-de-mongodb/
- https://www.slideshare.net/mongodb/webinar-exploring-the-aggregation-framework