

Bases de datos no relacionales

Recordar que utilizaremos de ejemplo mongodb.

Base de datos

La base de datos es un contenedor físico para las colecciones. Cada base de datos tiene su propio conjunto de ficheros en el sistema de ficheros.

Estructura mínima de almacenamiento

Colección

Una colección es un grupo de documentos. Es el equivalente a una tabla RDBMS. Una colección existe dentro de una única base de datos. Las colecciones no imponen un esquema. Los documentos de una colección pueden tener campos diferentes. Normalmente, todos los documentos de una colección tienen un propósito similar o relacionado.

Documento

Un documento es un conjunto de pares clave-valor. Los documentos tienen un esquema dinámico. Un esquema dinámico significa que los documentos de una misma colección no tienen por qué tener el mismo conjunto de campos o estructura, y que los campos comunes de los documentos de una colección pueden contener distintos tipos de datos.

Almacena en soportes informáticos una estructura lógica de almacenamiento, como la tiene un archivo de papel, por ejemplo: edificio, planta, pasillo, ubicación, ficha. De este modo es posible recuperar la información que interesa de un modo ágil, gracias a los índices y la estructura organizada del archivo.

Comparación

Base De Datos Relacional (SQL)	Base De Datos No Relacional (NOSQL)
Tabla	Colección
Fila	Documento
Columna	Campo clave
Relación entre tablas (clave foránea)	Documentos embebidos
Clave primaria	Clave primaria

Creación de una base de datos

USE

Crea nuevas bases de datos donde se van a almacenar las tablas.

```
use nombre_base_de_datos;
```

Vamos a ver el ejemplo de la creación de la base de datos tutorial.

Comando para ejecutar en consola:

```
use tutorial;
```

```
admin> use tutorial  
switched to db tutorial  
tutorial>
```

Mongo express:



Creación de colecciones

CREATE

Crea nuevas colecciones donde se van a almacenar los datos.

```
db.createCollection(nombre_de_la_colleccion);
```

Vamos a ver el ejemplo de la creación de la colección PROFESORES.

Comando para ejecutar en consola:

```
db.createCollection('PROFESORES');
```

```
tutorial> db.createCollection('PROFESORES');  
{ ok: 1 }  
tutorial>
```

Mongo express:

Viewing Database: tutorial

Collections

PROFESORES

+ Create collection

Viewing Collection: PROFESORES

Collection created!

New Document New Index

Simple Advanced

Key Value String Find

No documents found.

DROP

Elimina una colecciones.

```
db.nombre_de_la_colleccion.drop();
```

Vamos a ver el ejemplo de la creación de la colección PROFESORES.

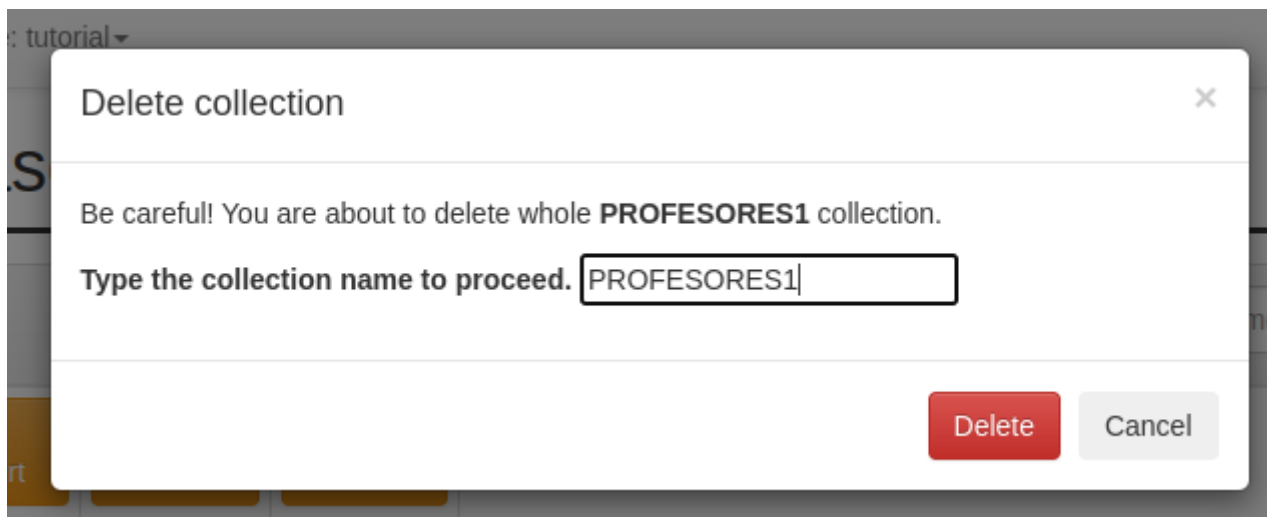
Comando para ejecutar en consola:

```
db.PROFESORES.drop();
```

```
tutorial> db.PROFESORES.drop();
true
tutorial>
```

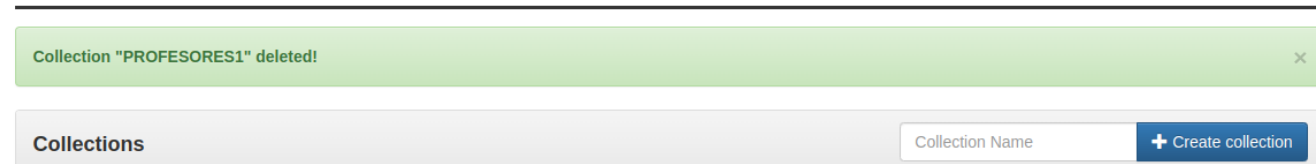
Mongo express:

View Export [JSON] Import PROFESORES1 Del



Mongo Express Database: tutorial

Viewing Database: tutorial



INSERT, UPDATE, DELETE SQL

Insert

La instrucción INSERT permite crear o insertar nuevos documentos en una colección. Recordar que por defecto se crea un campo llamado `_id` con un valor único alfa numérico.

Comando para ejecutar en consola:

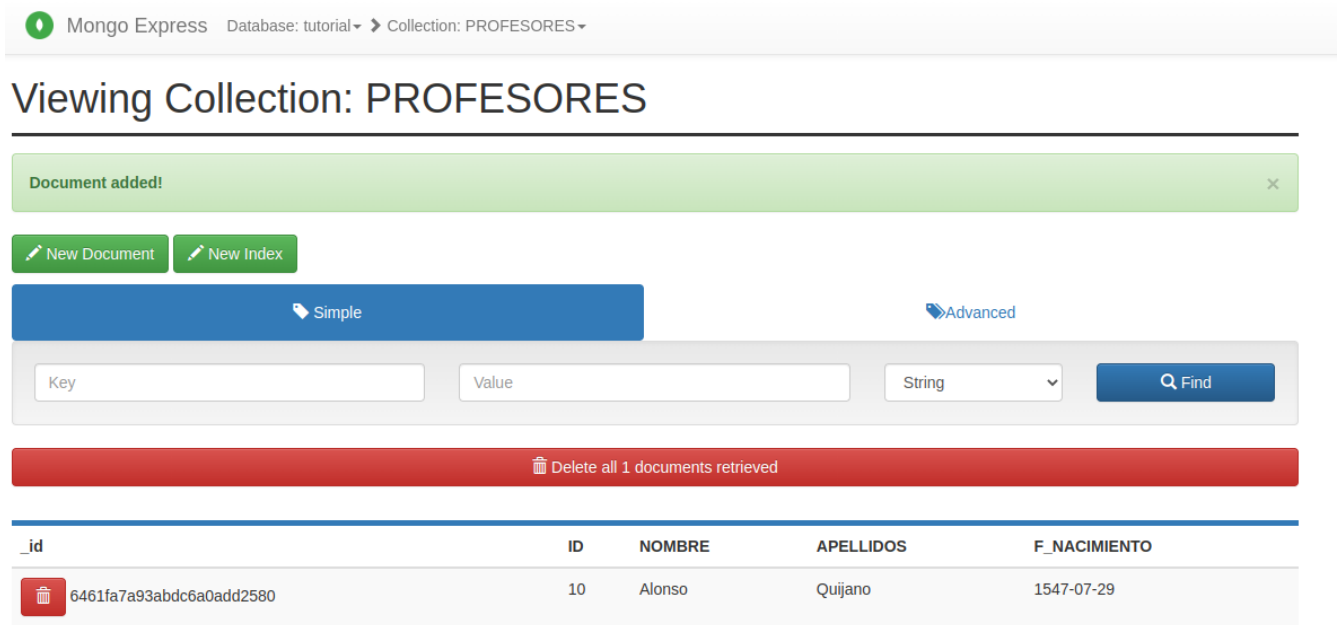
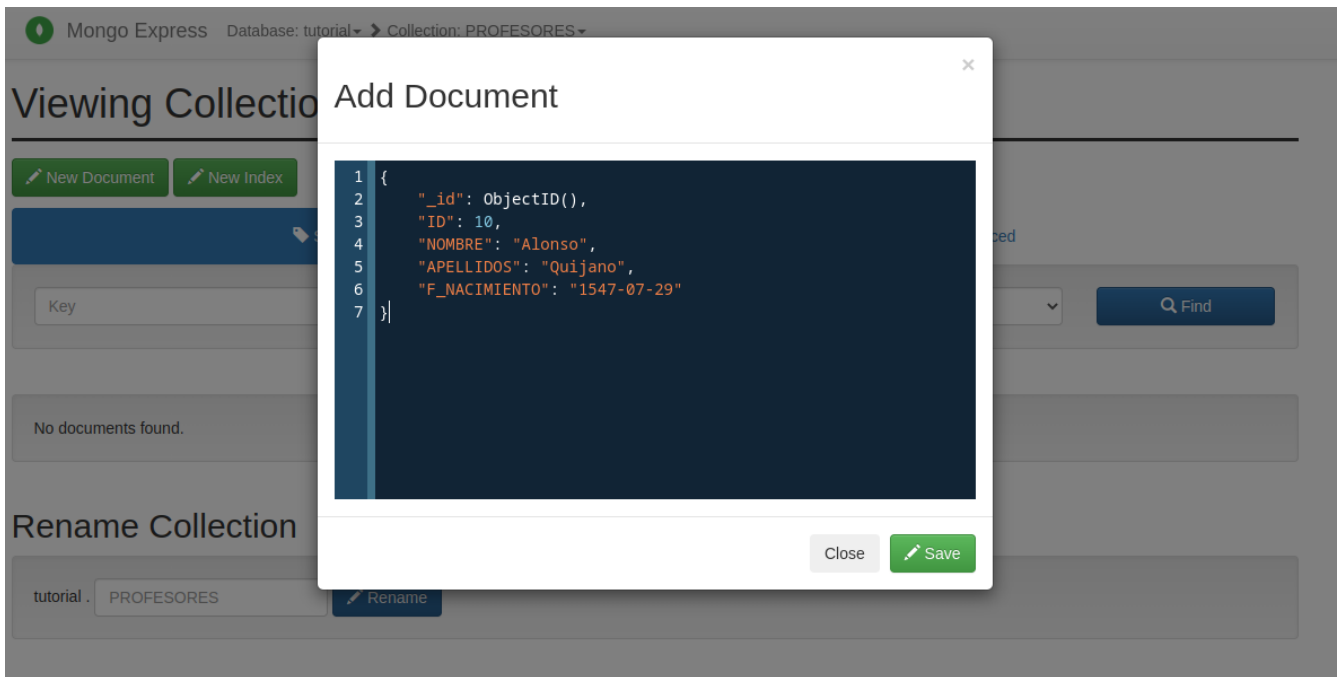
```
db.PROFESORES.insertOne({
  "ID": 10,
  "NOMBRE": "Alonso",
  "APELLIDOS": "Quijano",
  "F_NACIMIENTO": "1547-07-29"});
```

```
tutorial> db.PROFESORES.insertOne({
...   "ID": 10,
...   "NOMBRE": "Alonso",
...   "APELLIDOS": "Quijano",
...   "F_NACIMIENTO": "1547-07-29"});
{
  acknowledged: true,
  insertedId: ObjectId("6468e31a57a1f5a5a140fe8b")
}
```

El documento tiene formato json y lo podemos comparar con sql de la siguiente forma, cada clave del json es una columna y cada valor del json es el valor de la columna.

Mongo express:

Debemos presionar el botón de **New Document** en color verde.



Observar como el campo `_id` toma el valor **6461fa7a93abdc6a0add2580**.

Update SQL

La instrucción `UPDATE` permite actualizar documentos de una colección. Debemos por lo tanto indicar que documentos se quiere actualizar mediante el primer argumento de la función (`{"ID": 10}`), y que campos mediante el segundo argumento (`{$set: {"APELLIDOS": "Quijano (Don Quijote)"}}`)

Comando para ejecutar en consola:

```
db.PROSEFORES.updateOne(
  {"ID": 10},
  {$set: {"APELLIDOS": "Quijano (Don Quijote)}});
```

```
tutorial> db.PROSEFORES.updateOne(
...   {"ID": 10},
...   {$set: {"APELLIDOS": "Quijano (Don Quijote)}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
tutorial>
```

Mongo express:

Debemos hacer un click sobre el documento y modificamos los datos y finalmente presionamos **Save**.

Mongo Express Database: tutorial > Collection: PROFESORES > Document 6461fa7a93abdc6a0add2580

Editing Document: 6461fa7a93abdc6a0add2580

```
1 {
2   _id: ObjectId('6461fa7a93abdc6a0add2580'),
3   ID: 10,
4   NOMBRE: 'Alonso',
5   APELLIDOS: 'Quijano',
6   F_NACIMIENTO: '1547-07-29'
7 }
```

← Back

Save

Delete

Mongo Express Database: tutorial > Collection: PROFESORES

Viewing Collection: PROFESORES

Document updated!

New Document

New Index

Simple

Advanced

Key

Value

String

Find

Delete all 1 documents retrieved

_id	ID	NOMBRE	APELLIDOS	F_NACIMIENTO
 6461fa7a93abdc6a0add2580	10	Alonso	Quijano (Don Quijote)	1547-07-29

Delete SQL

La instrucción DELETE permite eliminar documentos de una colección, su sintaxis es simple, puesto que solo debemos indicar que registros deseamos eliminar mediante el primer argumento de la función (**{"ID": 10}**).

Comando para ejecutar en consola:


```
db.PROSEFORES.deleteOne({"ID": 10});
```

```
tutorial> db.PROSEFORES.deleteOne({"ID": 10});  
{ acknowledged: true, deletedCount: 0 }  
tutorial>
```

Mongo express:

Debemos presionar el botón delete en rojo.

The screenshot shows the Mongo Express web interface. At the top, it says "Mongo Express Database: tutorial Collection: PROFESORES". A modal dialog box is open in the center with the text "Are you sure?" and two buttons: "Delete" (in red) and "Cancel". Below the dialog, a green banner says "Document updated!". Underneath, there are buttons for "New Document" and "New Index". A search bar is visible with "Simple" and "Advanced" tabs. Below the search bar, a red banner says "Delete all 1 documents retrieved". At the bottom, a table displays the collection data:

_id	ID	NOMBRE	APELLIDOS	F_NACIMIENTO
 6461fa7a93abdc6a0add2580	10	Alonso	Quijano (Don Quijote)	1547-07-29

Viewing Collection: PROFESORES

Document deleted! _id: 6461fa7a93abdc6a0add2580

[New Document](#)[New Index](#)[Simple](#)[Advanced](#)[Find](#)

No documents found.

Rename Collection

[Rename](#)

Consultas SQL

Vamos a listar el nombre y los apellidos de los empleados que tienen un salario superior a 1350.

Comando para ejecutar en consola:

```
db.EMPLEADOS.find({SALARIO: {$gt:1350}}, {_id:0, NOMBRE: 1, APELLIDOS: 2})
```

```
tutorial> db.EMPLEADOS.find({SALARIO: {$gt:1350}}, {_id:0, NOMBRE: 1, APELLIDOS: 2})
[
  { NOMBRE: 'Carlos', APELLIDOS: 'Jiménez Clarín' },
  { NOMBRE: 'José', APELLIDOS: 'Calvo Sisman' }
]
tutorial>
```

Mongo express:

Viewing Collection: EMPLEADOS

[New Document](#) [New Index](#)

[Simple](#) [Advanced](#)

```
{SALARIO: {$gt:1350}}
```

```
{_id:0, NOMBRE: 1, APELLIDOS:2}
```

[Find](#)

Delete all 4 documents retrieved

_id	ID	NOMBRE	APELLIDOS	F_NACIMIENTO	SEXO	CARGO	SALARIO
6462b76393abdc6a0add2592	1	Carlos	Jiménez Clarín	1985-05-03	H	Mozo	1500
6462b76a93abdc6a0add2594	2	Elena	Rubio Cuestas	1978-09-25	M	Secretaria	1300
6462b77293abdc6a0add2596	3	José	Calvo Sisman	1990-11-12	H	Mozo	1400
6462b77b93abdc6a0add2598	4	Margarita	Rodríguez Garcés	1992-05-16	M	Secretaria	1325.5

Viewing Collection: EMPLEADOS

[New Document](#) [New Index](#)

[Simple](#) [Advanced](#)

```
{SALARIO: {$gt:1350}}
```

```
{_id:0, NOMBRE: 1, APELLIDOS:2}
```

[Find](#)

Delete all 2 documents retrieved

NOMBRE	APELLIDOS
Carlos	Jiménez Clarín
José	Calvo Sisman

Rename Collection

tutorial . EMPLEADOS [Rename](#)

En este caso la función **find** recibe como primer argumento el filtro y como segundo parámetro las columnas que deseo obtener como respuesta.

Filtro (query): {SALARIO: {\$gt:1350}}

En este caso estamos filtrando la key o columna **SALARIO** para que sea mayor a 1350.

Campos retornados (projection): {_id:0, NOMBRE: 1, APELLIDOS:2}

Debemos el nombre de las keys (columnas) y la posición en que se van a mostrar. En caso de querer omitir el campo por defecto **_id** se lo debemos pasar en la posición 0 como muestra el ejemplo.

Modificando el filtro


Vamos a modificar el filtro para que nos muestre los empleados que tienen un salario entre 1350 y 1450.

Comando para ejecutar en consola:

```
db.EMPLEADOS.find({$and:[{SALARIO: {$lt:1450}},{SALARIO: {$gt:1350}}]}, {_id:0, NOMBRE: 1, APELLIDOS: 2})
```

```
tutorial> db.EMPLEADOS.find({$and:[{SALARIO: {$lt:1450}},{SALARIO: {$gt:1350}}]}, {_id:0, NOMBRE: 1, APELLIDOS: 2})
[ { NOMBRE: 'Jose', APELLIDOS: 'Calvo Sisman' } ]
tutorial>
tutorial>
```

Mongo express:

 Mongo Express Database: tutorial ▶ Collection: EMPLEADOS ▼

Viewing Collection: EMPLEADOS

New Document New Index

Simple





Advanced

{ \$and: [{ SALARIO: { \$lt: 1450 } }, { SALARIO: { \$gt: 1350 } }] }

{ _id: 0, NOMBRE: 1, APELLIDOS: 2 }

Find

Delete all 4 documents retrieved

_id	ID	NOMBRE	APELLIDOS	F_NACIMIENTO	SEXO	CARGO	SALARIO
 6462b76393abdc6a0add2592	1	Carlos	Jiménez Clarín	1985-05-03	H	Mozo	1500
 6462b76a93abdc6a0add2594	2	Elena	Rubio Cuestas	1978-09-25	M	Secretaria	1300
 6462b77293abdc6a0add2596	3	José	Calvo Sisman	1990-11-12	H	Mozo	1400
 6462b77b93abdc6a0add2598	4	Margarita	Rodríguez Garcés	1992-05-16	M	Secretaria	1325.5

Viewing Collection: EMPLEADOS

[New Document](#) [New Index](#)

[Simple](#) [Advanced](#)

```
{ $and: [{ SALARIO: { $lt: 1450 } }, { SALARIO: { $gt: 1350 } } ] }
```

```
{ _id: 0, NOMBRE: 1, APELLIDOS: 2 }
```

[Find](#)

Delete all 1 documents retrieved

NOMBRE	APELLIDOS
José	Calvo Sisman

Rename Collection

tutorial . [Rename](#)

Tipos de dato

Son los tipos de datos de javascript:

- string: Las cadenas de texto. Se utilizan con comillas dobles.
- int o long: Para representar números enteros y no llevan comillas dobles.
- decimal: Para representar números decimales y no llevan comillas dobles.
- date: Para representar fechas.
- datetime: Para representar fecha y hora.
- array: Arreglos de datos. Se utilizan los corchetes para representar un lista de valores o objetos.
- object: Objetos. Se utilizan las llaves.

IMPORTANT

Recordar que se trabajan con objetos javascript o json

Operadores

Es un operador que opera normalmente entre dos operandos, estableciendo una operación que al ejecutarla se obtiene un resultado.

Lógica booleana

Nos permite establecer condiciones que pueden ser verdaderas o falsas.

Expresiones booleanas

En la consulta

```
db.EMPLEADOS.find({SALARIO: {$gt:1350}}, {_id:0, NOMBRE: 1, APELLIDOS: 2})
```

```
tutorial> db.EMPLEADOS.find({SALARIO: {$gt:1350}}, {_id:0, NOMBRE: 1, APELLIDOS: 2})
[
  { NOMBRE: 'Carlos', APELLIDOS: 'Jiménez Clarín' },
  { NOMBRE: 'José', APELLIDOS: 'Calvo Sisman' }
]
tutorial>
tutorial>
```

dentro del filtro `SALARIO > 1350`, estamos estableciendo una expresión booleana donde ">" es el operador, "SALARIO" es un operando variable, que tomará valores de cada registro de la tabla EMPLEADOS, y "1350" es un operando constante. El resultado de esta expresión depende del valor que tome la variable SALARIO, pero en cualquier caso sólo puede dar dos posibles resultados, verdadero o falso.

Operadores

Símbolo	Descripción
\$eq	Igual a
\$gt	Mayor a
\$lt	Menor a
\$gte	Mayor o igual a
\$lte	Menor o igual a

Operadores lógicos

Símbolo	Descripción
\$and	Operador y
\$or	Operador o
\$not	Operador no
\$nor	Operador no

Table 1. Tabla de verdad para el operador lógico NOT

A	Not A
true	false
false	true

Table 2. Tabla de verdad para el operador lógico AND

A	B	A AND B
true	true	true

A	B	A AND B
true	false	false
false	false	false
false	true	false

Table 3. Tabla de verdad para el operador lógico OR

A	B	A OR B
true	true	true
true	false	true
false	false	false
false	true	true

Table 4. Tabla de verdad para el operador lógico NOR

A	B	A NOR B
true	true	false
true	false	false
false	false	true
false	true	false

El operador AND

¿qué personas son rubias y altas?, para ello construimos la siguiente consulta SQL:

Comando para ejecutar en consola:

```
db.PERSONAS.find({$and: [{RUBIA: "S"}, {ALTA: "S"}]}, {_id: 0, NOMBRE: 1})
// o
db.PERSONAS.find({$and: [{RUBIA: {$eq: "S"}}, {ALTA: {$eq: "S"}}]}, {_id: 0, NOMBRE: 1})
```

```
tutorial> db.PERSONAS.find({$and: [{RUBIA: "S"}, {ALTA: "S"}]}, {_id: 0, NOMBRE: 1})
[ { NOMBRE: 'Manuel' }, { NOMBRE: 'José' } ]
tutorial>
tutorial> □
```

Mongo express:

Viewing Collection: PERSONAS

New Document

New Index

Simple

Advanced

{ \$and: [{ RUBIA: "S" }, { ALTA: "S" }] }

{ _id: 0, NOMBRE: 1 }

Find

Delete all 5 documents retrieved

_id	ID	NOMBRE	RUBIA	ALTA	GAFAS
6462b8a593abdc6a0add25da	1	Manuel	S	S	N
6462b8ab93abdc6a0add25dc	2	Maria	N	N	S
6462b8b193abdc6a0add25de	3	Carmen	S	N	S
6462b8b793abdc6a0add25e0	4	José	S	S	S
6462b8bd93abdc6a0add25e2	5	Pedro	N	S	N

Viewing Collection: PERSONAS

New Document

New Index

Simple

Advanced

{ \$and: [{ RUBIA: "S" }, { ALTA: "S" }] }

{ _id: 0, NOMBRE: 1 }

Find

Delete all 2 documents retrieved

NOMBRE
Manuel
José

Rename Collection

tutorial . PERSONAS

Rename

El operador OR


Supongamos que queremos saber las personas que son rubias o bien altas, es decir, queremos que si es rubia la considere con independencia de su altura, y a la inversa, también queremos que la seleccione si es alta independientemente del color de pelo. La consulta sería la siguiente.

Comando para ejecutar en consola:

```
db.PERSONAS.find({$or: [{RUBIA: "S"}, {ALTA: "S"}]}, {_id: 0, NOMBRE: 1})
// o
db.PERSONAS.find({$or: [{RUBIA: {$eq: "S"}}, {ALTA: {$eq: "S"}}]}, {_id: 0, NOMBRE: 1})
```

```
tutorial> db.PERSONAS.find({$or: [{RUBIA: "S"}, {ALTA: "S"}]}, {_id: 0, NOMBRE: 1})
[
  { NOMBRE: 'Manuel' },
  { NOMBRE: 'Carmen' },
  { NOMBRE: 'José' },
  { NOMBRE: 'Pedro' }
]
tutorial>
tutorial>
```

Mongo express:

 Mongo Express Database: tutorial ▶ Collection: PERSONAS ▶

Viewing Collection: PERSONAS

New Document

New Index

Simple






Advanced

{ \$or: [{ RUBIA: "S" }, { ALTA: "S" }] }

{ _id: 0, NOMBRE: 1 }

Find

Delete all 5 documents retrieved

_id	ID	NOMBRE	RUBIA	ALTA	GAFAS
 6462b8a593abdc6a0add25da	1	Manuel	S	S	N
 6462b8ab93abdc6a0add25dc	2	Maria	N	N	S
 6462b8b193abdc6a0add25de	3	Carmen	S	N	S
 6462b8b793abdc6a0add25e0	4	José	S	S	S
 6462b8bd93abdc6a0add25e2	5	Pedro	N	S	N

Viewing Collection: PERSONAS

New Document
New Index

Simple
Advanced

```
{ $or: [{ RUBIA: "S" }, { ALTA: "S" }] }
```

```
{ _id: 0, NOMBRE: 1 }
```

Find

Delete all 5 documents retrieved

_id	ID	NOMBRE	RUBIA	ALTA	GAFAS
6462b8a593abdc6a0add25da	1	Manuel	S	S	N
6462b8ab93abdc6a0add25dc	2	Maria	N	N	S
6462b8b193abdc6a0add25de	3	Carmen	S	N	S
6462b8b793abdc6a0add25e0	4	José	S	S	S
6462b8bd93abdc6a0add25e2	5	Pedro	N	S	N

El operador NOT

Este operador tan solo tiene un operando, el resultado es negar el valor del operando.

Tomemos la anterior consulta y neguemos el filtro, si antes el resultado era: **Manuel, Carmen, José y Pedro** ahora el resultado ha de ser **María**.

Comando para ejecutar en consola:

```
db.PERSONAS.find({ $nor: [{ RUBIA: "S" }, { ALTA: "S" } ] }, { _id: 0, NOMBRE: 1 })
```

```
tutorial> db.PERSONAS.find({ $nor: [{ RUBIA: "S" }, { ALTA: "S" } ] }, { _id: 0, NOMBRE: 1 })
[ { NOMBRE: 'Maria' } ]
tutorial>
tutorial>
```

Mongo express:

Viewing Collection: PERSONAS

New Document

New Index

Simple

Advanced

```
{ $nor: [{ RUBIA: "S" }, { ALTA: "S" } ] }
```

```
{ _id: 0, NOMBRE: 1 }
```

Find

Delete all 5 documents retrieved

_id	ID	NOMBRE	RUBIA	ALTA	GAFAS
 6462b8a593abdc6a0add25da	1	Manuel	S	S	N
 6462b8ab93abdc6a0add25dc	2	Maria	N	N	S
 6462b8b193abdc6a0add25de	3	Carmen	S	N	S
 6462b8b793abdc6a0add25e0	4	José	S	S	S
 6462b8bd93abdc6a0add25e2	5	Pedro	N	S	N

Viewing Collection: PERSONAS

New Document

New Index

Simple

Advanced

```
{ $nor: [{ RUBIA: "S" }, { ALTA: "S" } ] }
```

```
{ _id: 0, NOMBRE: 1 }
```

Find

Delete all 1 documents retrieved

NOMBRE
Maria

Rename Collection

tutorial . PERSONAS

Rename

Tools

Totalizar datos

Para este tipo de funciones utilizaremos las agregaciones.

¿Cuál es el salario medio de los empleados?

Comando para ejecutar en consola:

```
db.EMPLEADOS.aggregate([{$group: {_id: null, totalSalario: { $sum: "$SALARIO" }}}},
{$project: {_id:0, totalSalario: 1 }}}])
```

```
tutorial> db.EMPLEADOS.aggregate([{$group: {_id: null, totalSalario: { $sum: "$SALARIO" }}}}, {$project: {_id:0, totalSalario: 1 }}}])
[ { totalSalario: 5525.5 } ]
tutorial>
tutorial>
```

Dentro de la función aggregate, utilizamos el operador \$group para agrupar todos los documentos en una sola salida. El campo _id se establece en null para agrupar todos los documentos sin considerar ningún campo específico. Utilizamos el operador de acumulación \$sum para sumar los valores del campo SALARIO. El resultado de la suma se guarda en un nuevo campo llamado totalSalario.

Fíjese que el resultado de esta consulta devuelve una sola clave y un valor.

WARNING

No todas las consultas funcionan por la aplicación mongo-express.

Análogamente contamos el número de empleados, es decir, el número de registros de la tabla empleados.

```
db.EMPLEADOS.aggregate([{$group: {_id: null, totalEmpleados: { $sum: 1 }}}}, {$project:
{_id:0, totalEmpleados: 1 }}}])
```

```
tutorial> db.EMPLEADOS.aggregate([{$group: {_id: null, totalEmpleados: { $sum: 1 }}}}, {$project: {_id:0, totalEmpleados: 1 }}}])
[ { totalEmpleados: 4 } ]
tutorial>
tutorial>
```

Notar como en este caso en lugar de sumar la clave \$SALARIO estamos sumando un valor fijo 1.

Ahora ya podemos resolver la cuestión planteada, basta con dividir el primer resultado por el segundo.

```
db.EMPLEADOS.aggregate([{$group: {_id: null, totalEmpleados: { $sum: 1 }},
totalSalario: { $sum: "$SALARIO" }}}}, {$project: {_id:0, salarioPromedio: {$divide:
["$totalSalario", "$totalEmpleados"] }}}])
```

```
tutorial> db.EMPLEADOS.aggregate([{$group: {_id: null, totalEmpleados: { $sum: 1 }}}}, {$project: {_id:0, totalEmpleados: 1 }}}])
[ { totalEmpleados: 4 } ]
tutorial>
tutorial>
```

Los totalizadores utilizan pipelines para trabajar, es decir, primero se ejecuta una etapa, luego otra

etapa y así sucesivamente. La consulta anterior tiene 2 etapas:

1. `{ $group: { _id: null, totalEmpleados: { $sum: 1 }, totalSalario: { $sum: "$SALARIO" } }}`: Etapa donde se totalizan los valores de **totalSalario** y **totalEmpleados**, estas 2 nuevas salidas son claves y valores que se pasan a la siguiente etapa.
2. `{ $project: { _id: 0, salarioPromedio: { $divide: ["$totalSalario", "$totalEmpleados"] } }}`: En esta etapa vamos a mostrar el resultado con la orden **\$projection**, dentro de la proyección estamos utilizando la orden **\$divide** para dividir 2 valores que en este caso son los valores que se obtuvieron en la etapa anterior.

Agrupación de datos (aggregate)

Para las agregaciones vamos a trabajar con los pipelines de mongodb, es decir, vamos a aplicar distintas acciones a la hora de realizar una consulta.

Cláusula GROUP BY \$group

¿cuantos empleados de cada sexo hay?

```
db.EMPLEADOS.aggregate([{$group: {_id: "$SEXO", EMPLEADOS: { $sum: 1 } }}, {$project:
{_id: 0, KEY_SEXO: "$_id", KEY_EMPLEADOS: "$EMPLEADOS"} }])
```

```
tutorial> db.EMPLEADOS.aggregate([{$group: {_id: null, totalEmpleados: { $sum: 1 }, totalSalario: { $sum: "$SALARIO" } }}, {$project: {_id: 0, salarioPromedio: { $divide: ["$totalSalario", "$totalEmpleados"] } } }])
[ { salarioPromedio: 1381.375 } ]
tutorial>
```

Observe que el resultado de la consulta devuelve dos objetos. Para realizar un cambio de nombre utilizamos el **\$project**.

Etapas:

1. `{ $group: { _id: "$SEXO", EMPLEADOS: { $sum: 1 } }}`: agrupar por el **_id** que toma el valor de la key (columna) SEXO y se suma sobre una key llamada EMPLEADO.
2. `{ $project: { _id: 0, SEXO: "$_id", EMPLEADOS: 1 } }`: aquí se produce un renombre de campos, **_id: 0** indica que no se muestra el campo **_id**, luego la key KEY_SEXO la voy a tomar del valor del filtro **\$_id** y el valor de la key KEY_EMPLEADOS la voy a tomar del filtro **\$EMPLEADOS**.

En este caso el pipeline se construye de un filtro y de una proyección, la cual me permite renombrar las columnas.

La palabra clave DISTINCT

Con ella podemos eliminar filas redundantes de un resultado SQL, por lo que permite obtener los distintos valores de un campo existentes en una tabla o grupo de registros seleccionados.

Por ejemplo, ¿qué valores distintos existen en el campo SEXO de la tabla empleados?:

```
db.EMPLEADOS.aggregate([{$group: { _id: "$SEXO" } }])
```

```
tutorial>
tutorial> db.EMPLEADOS.aggregate([{$group: { _id: "$SEXO" } }])
[ { _id: 'H' }, { _id: 'M' } ]
tutorial>
tutorial>
```

Utilizaremos la colección MASCOTAS:

¿cuantos perros de cada sexo hay en total actualmente en el centro?

Consulta:

```
db.MASCOTAS.aggregate([{$match: {ESPECIE: 'P',ESTADO: 'A'}}, {$group: {_id: '$SEXO',
PERROS_VIGENTES: { $sum: 1 }}}])
```

```
tutorial>
tutorial> db.MASCOTAS.aggregate([{$match: {ESPECIE: 'P',ESTADO: 'A'}}, {$group: {_id: '$SEXO', PERROS_VIGENTES: { $sum: 1 }}}])
[ { _id: 'H', PERROS_VIGENTES: 5 }, { _id: 'M', PERROS_VIGENTES: 2 } ]
tutorial>
tutorial>
```

El resultado son dos machos y cinco hembras.

En este caso utilizamos la instrucción **\$match** para hacer que se cumplan las condiciones de ESPECIE y ESTADO (también lo podemos realizar con un **\$and**) y luego agrupamos por la key SEXO.

Filtrar cálculos de totalización (SQL HAVING)

Para aplicar los SQL HAVING se aplican los pipelines nuevamente.

Cláusula HAVING

¿Qué ubicaciones del centro de mascotas tienen más de dos ejemplares?

Consulta SQL:

```
db.MASCOTAS.aggregate([{$match: { ESTADO: 'A' } },{$group: { _id: "$UBICACION",
EJEMPLARES: { $sum: 1 } } },{$match: { EJEMPLARES: { $gt: 2 } } }])
```

```
tutorial> db.MASCOTAS.aggregate([{$match: { ESTADO: 'A' } },{$group: { _id: "$UBICACION", EJEMPLARES: { $sum: 1 } } },{$match: { EJEMPLARES: { $gt: 2 } } }])
[ { _id: 'E04', EJEMPLARES: 3 }, { _id: 'E02', EJEMPLARES: 4 } ]
tutorial>
tutorial>
```

En este caso primero se filtran todos los documentos que poseen el ESTADO igual a A, luego agrupo los documentos por la key UBICACION y por último aplico un nuevo filtro donde la key EJEMPLARES sea mayor a 2.

Ordenación del resultado (SQL ORDER BY)

La Cláusula ORDER BY nos permite ordenar las filas de resultado por una o más columnas. Esta cláusula no se presenta en última instancia por casualidad, sino por que siempre irá al final de una consulta o sea antes de devolver el resultado.

Una última cláusula implica una última pregunta de construcción:
¿Cómo deben ordenarse los datos resultantes?

Supongamos que queremos obtener una lista ordenada de los empleados por sueldo, de modo que primero este situado el de menor salario y por último el de mayor:

```
db.EMPLEADOS.find({}, {_id:0, NOMBRE: 1, APELLIDOS: 1, SALARIO: 1 }).sort({ SALARIO: 1
})
// o
db.EMPLEADOS.aggregate([{$sort: {SALARIO: 1}}, {$project: {_id:0, NOMBRE: 1,
APELLIDOS: 2, SALARIO: 3 }}}])
```

```
tutorial> db.EMPLEADOS.aggregate([{$sort: {SALARIO: 1}}, {$project: {_id:0, NOMBRE: 1, APELLIDOS: 2, SALARIO: 3 }}}])
[
  { NOMBRE: 'Elena', APELLIDOS: 'Rubio Cuestas', SALARIO: 1300 },
  {
    NOMBRE: 'Margarita',
    APELLIDOS: 'Rodríguez Garcés',
    SALARIO: 1325.5
  },
  { NOMBRE: 'José', APELLIDOS: 'Calvo Sisman', SALARIO: 1400 },
  { NOMBRE: 'Carlos', APELLIDOS: 'Jiménez Clarín', SALARIO: 1500 }
]
tutorial>
tutorial>
```

En este caso vemos que podemos utilizar la función **sort** con el campo/s que deseamos utilizar para ordenar el listado. En este caso **SALARIO: 1** indica que se ordena de forma ascendente y si ponemos **SALARIO: -1** se ordena en forma descendente.

En este caso también puedo usar el aggregate con la orden **\$sort** para poder utilizar el pipeline y obtener el mismo resultado.

```
db.EMPLEADOS.find({}, {_id:0, NOMBRE: 1, APELLIDOS: 1, SALARIO: 1 }).sort({ SALARIO:
-1 })
// o
db.EMPLEADOS.aggregate([{$sort: {SALARIO: -1}}, {$project: {_id:0, NOMBRE: 1,
APELLIDOS: 2, SALARIO: 3 }}}])
```

```
tutorial> db.EMPLEADOS.aggregate([{$sort: {SALARIO: -1}}, {$project: {_id:0, NOMBRE: 1, APELLIDOS: 2, SALARIO: 3 }}}])
[
  { NOMBRE: 'Carlos', APELLIDOS: 'Jiménez Clarín', SALARIO: 1500 },
  { NOMBRE: 'José', APELLIDOS: 'Calvo Sisman', SALARIO: 1400 },
  {
    NOMBRE: 'Margarita',
    APELLIDOS: 'Rodríguez Garcés',
    SALARIO: 1325.5
  },
  { NOMBRE: 'Elena', APELLIDOS: 'Rubio Cuestas', SALARIO: 1300 }
]
tutorial>
tutorial>
```

La orden \$regex (LIKE) / El valor NULL

La orden \$regex

En esta caso la orden utiliza expresiones regulares y está limitada por el caracter /, es decir, que una expresión regular comienza y termina con /.

¿Qué empleados su primer apellido comienza por "R"? Veamos primero la consulta SQL que responde a esto:

```
db.EMPLEADOS.find({ APELLIDOS: {$regex: /^R/ } })
//0
db.EMPLEADOS.find({ APELLIDOS: /^R/ })
```

```
tutorial> db.EMPLEADOS.find({ APELLIDOS: /^R/ })
[
  {
    _id: ObjectId("6462b76a93abdc6a0add2594"),
    ID: 2,
    NOMBRE: 'Elena',
    APELLIDOS: 'Rubio Cuestas',
    F_NACIMIENTO: '1978-09-25',
    SEXO: 'M',
    CARGO: 'Secretaria',
    SALARIO: 1300
  },
  {
    _id: ObjectId("6462b77b93abdc6a0add2598"),
    ID: 4,
    NOMBRE: 'Margarita',
    APELLIDOS: 'Rodríguez Garcés',
    F_NACIMIENTO: '1992-05-16',
    SEXO: 'M',
    CARGO: 'Secretaria',
    SALARIO: 1325.5
  }
]
tutorial>
tutorial>
```

```
db.EMPLEADOS.find({ APELLIDOS: {$regex: /N$/ig } })
// 0
db.EMPLEADOS.find({ APELLIDOS: /N$/ig })
```

```
tutorial> db.EMPLEADOS.find({ APELLIDOS: /N$/ig })
[
  {
    _id: ObjectId("6462b76393abdc6a0add2592"),
    ID: 1,
    NOMBRE: 'Carlos',
    APELLIDOS: 'Jiménez Clarín',
    F_NACIMIENTO: '1985-05-03',
    SEXO: 'H',
    CARGO: 'Mozo',
    SALARIO: 1500
  },
  {
    _id: ObjectId("6462b77293abdc6a0add2596"),
    ID: 3,
    NOMBRE: 'José',
    APELLIDOS: 'Calvo Sisman',
    F_NACIMIENTO: '1990-11-12',
    SEXO: 'H',
    CARGO: 'Mozo',
    SALARIO: 1400
  }
]
tutorial>
tutorial>
```

NOTE En este caso con ig hacemos que el texto sea key insensitive.

Veamos una última aplicación de este recurso.

¿Qué devuelve esta consulta?:

```
db.EMPLEADOS.find({ APELLIDOS: {$regex: /.*AR./ig }})
// 0
db.EMPLEADOS.find({ APELLIDOS: /.*AR./ig })
// 0
db.EMPLEADOS.find({ APELLIDOS: /AR/ig })
```

```
tutorial> db.EMPLEADOS.find({}, {_id: 0, RETENCION: {$multiply: ["$SALARIO", 0.035]}, RETENCION_ROUND: {$round: [{$multiply: ["$SALARIO", 0.035]}, 2] }})
[
  { RETENCION: 52.50000000000001, RETENCION_ROUND: 52.5 },
  { RETENCION: 45.50000000000001, RETENCION_ROUND: 45.5 },
  { RETENCION: 49.00000000000001, RETENCION_ROUND: 49 },
  { RETENCION: 46.39250000000005, RETENCION_ROUND: 46.39 }
]
tutorial>
```

Funciones

CONCAT

Realiza la concatenación de dos o más cadenas de texto. Para este caso particular la orden **\$concat** solo está disponible para el aggregate y recibe un array con los campos/strings a concatenar.

```
db.EMPLEADOS.aggregate([{$project: {_id: 0, NOMBRE_APELLIDOS: { $concat: ["$NOMBRE", " ", "$APELLIDOS"] }}}])
```

```
tutorial> db.EMPLEADOS.find({ APELLIDOS: /AR/ig })
[
  {
    _id: ObjectId("6462b76393abdc6a0add2592"),
    ID: 1,
    NOMBRE: 'Carlos',
    APELLIDOS: 'Jiménez Clarín',
    F_NACIMIENTO: '1985-05-03',
    SEXO: 'M',
    CARGO: 'Mozo',
    SALARIO: 1500
  },
  {
    _id: ObjectId("6462b77b93abdc6a0add2598"),
    ID: 4,
    NOMBRE: 'Margarita',
    APELLIDOS: 'Rodríguez Garcés',
    F_NACIMIENTO: '1992-05-16',
    SEXO: 'M',
    CARGO: 'Secretaria',
    SALARIO: 1325.5
  }
]
tutorial>
```

CURRENT_DATE

Retorna la fecha del servidor.

```
db.EMPLEADOS.findOne({}, { localtime: { $dateToString: { format: "%Y-%m-%d %H:%M:%S", date: new Date() } } })
```

```
tutorial> db.EMPLEADOS.findOne({}, { localtime: { $dateToString: { format: "%Y-%m-%d %H:%M:%S", date: new Date() } } })
{
  _id: ObjectId("6462b76393abdc6a0add2592"),
  localtime: '2023-05-20 15:02:03'
}
tutorial>
```

En este ejemplo estamos creando un objeto de tipo **Date** que tiene la fecha y hora del servidor y

luego con la orden **\$dateToString** le podemos dar formato.

```
db.EMPLEADOS.find({}, { ID: 1, NOMBRE: 1, APELLIDOS: 1, F_NACIMIENTO: 1,
F_NACIMIENTO_FORMATEADA_DESDE_STRING: { $dateFromString: { format: "%Y-%m-%d",
dateString: "$F_NACIMIENTO" } }, F_NACIMIENTO_PASADA_A_DATE_Y_FORMATEADA_A_STRING: {
$dateToString: {format: "%d-%m-%Y", date: { $dateFromString: { format: "%Y-%m-%d",
dateString: "$F_NACIMIENTO" } } } } })
```

```
tutorial> db.EMPLEADOS.find({}, { ID: 1, NOMBRE: 1, APELLIDOS: 1, F_NACIMIENTO: 1, F_NACIMIENTO_FORMATEADA_DESDE_STRING: { $dateFromString: { format: "%Y-%m-%d", dateString: "$F_NACIMIENTO" } }, F_NACIMIENTO_PASADA_A_DATE_Y_FORMATEADA_A_STRING: { $dateToString: {format: "%d-%m-%Y", date: { $dateFromString: { format: "%Y-%m-%d", dateString: "$F_NACIMIENTO" } } } } })
[
  {
    _id: ObjectId("6462b76393abdc6a0add2592"),
    ID: 1,
    NOMBRE: 'Carlos',
    APELLIDOS: 'Jiménez Clarín',
    F_NACIMIENTO: '1985-05-03',
    F_NACIMIENTO_FORMATEADA_DESDE_STRING: ISODate("1985-05-03T00:00:00.000Z"),
    F_NACIMIENTO_PASADA_A_DATE_Y_FORMATEADA_A_STRING: '03-05-1985'
  },
  {
    _id: ObjectId("6462b76a93abdc6a0add2594"),
    ID: 2,
    NOMBRE: 'Elena',
    APELLIDOS: 'Rubio Cuestas',
    F_NACIMIENTO: '1978-09-25',
    F_NACIMIENTO_FORMATEADA_DESDE_STRING: ISODate("1978-09-25T00:00:00.000Z"),
    F_NACIMIENTO_PASADA_A_DATE_Y_FORMATEADA_A_STRING: '25-09-1978'
  },
  {
    _id: ObjectId("6462b77293abdc6a0add2596"),
    ID: 3,
    NOMBRE: 'José',
    APELLIDOS: 'Calvo Sisman',
    F_NACIMIENTO: '1990-11-12',
    F_NACIMIENTO_FORMATEADA_DESDE_STRING: ISODate("1990-11-12T00:00:00.000Z"),
    F_NACIMIENTO_PASADA_A_DATE_Y_FORMATEADA_A_STRING: '12-11-1990'
  },
  {
    _id: ObjectId("6462b77b93abdc6a0add2598"),
    ID: 4,
    NOMBRE: 'Margarita',
    APELLIDOS: 'Rodríguez Garcés',
    F_NACIMIENTO: '1992-05-16',
    F_NACIMIENTO_FORMATEADA_DESDE_STRING: ISODate("1992-05-16T00:00:00.000Z"),
    F_NACIMIENTO_PASADA_A_DATE_Y_FORMATEADA_A_STRING: '16-05-1992'
  }
]
```

En este último ejemplo vemos como pasar un string que almacena una fecha a otro string con otro formato. La orden **\$dateToString** pasa un objeto de tipo **Date** a un string y la key formato responde al formato que queremos que tenga la fecha, mientras que la orden **\$dateFromString** pasa un string a un objeto del tipo **Date** y la key formato responde al formato que tiene el string.

DATE_ADD / DATE_SUB

Se utiliza para agregar / quitar valores a las fechas. Como parámetros recibe la fecha y el intervalo de valor. Se pueden agregar días, meses, años, horas, minutos.... Los intervalos pueden variar según el motor de base de datos.

```
db.EMPLEADOS.findOne({}, {
  FECHA_ACTUAL_MAS_TREINTA_DIAS: { $add: [new Date(), { $multiply: [30, 24, 60, 60, 1000] } ] },
  FECHA_ACTUAL_MAS_SEIS_MESES: { $add: [new Date(), { $multiply: [6, 30, 24, 60, 60, 1000] } ] },
  FECHA_ACTUAL_MENOS_TREINTA_DIAS: { $subtract: [new Date(), { $multiply: [30, 24, 60, 60, 1000] } ] },
  FECHA_ACTUAL_MENOS_SEIS_MESES: { $subtract: [new Date(), { $multiply: [6, 30, 24, 60, 60, 1000] } ] }
})
```



```
tutorial> db.EMPLEADOS.findOne({}, {
...   FECHA_ACTUAL_MAS_TREINTA_DIAS: { $add: [new Date(), { $multiply: [30, 24, 60, 60, 1000] }] },
...   FECHA_ACTUAL_MAS_SEIS_MESES: { $add: [new Date(), { $multiply: [6, 30, 24, 60, 60, 1000] }] },
...   FECHA_ACTUAL_MENOS_TREINTA_DIAS: { $subtract: [new Date(), { $multiply: [30, 24, 60, 60, 1000] }] },
...   FECHA_ACTUAL_MENOS_SEIS_MESES: { $subtract: [new Date(), { $multiply: [6, 30, 24, 60, 60, 1000] }] }
... })
{
  _id: ObjectId("6462b76393abdc6a0add2592"),
  FECHA_ACTUAL_MAS_TREINTA_DIAS: ISODate("2023-06-19T15:03:21.921Z"),
  FECHA_ACTUAL_MAS_SEIS_MESES: ISODate("2023-11-16T15:03:21.921Z"),
  FECHA_ACTUAL_MENOS_TREINTA_DIAS: ISODate("2023-04-20T15:03:21.921Z"),
  FECHA_ACTUAL_MENOS_SEIS_MESES: ISODate("2022-11-21T15:03:21.921Z")
}
```

En este caso debemos utilizar las ordenes **\$add**, **\$subtract** y **\$multiply** para sumar, restar o multiplicar valores. Al objeto **Date** le tenemos que sumar o restar el valor en milisegundos es por eso la lista de valores, por ejemplo [30 (días), 24 (horas), 60 (minutos), 60 (segundos), 1000 (milisegundos)], todos estos valores se multiplican y luego se suman al Date.

SUBSTR

Retorna el substring de una cadena. Como parámetros recibe el dato de tipo cadena a tratar en primer lugar, seguido de la posición dentro de la cadena donde se quiere obtener la subcadena, y por último la longitud o número de caracteres de esta. Ejemplos:

```
db.EMPLEADOS.findOne({}, {_id: 0, Nombre: 1, NOMBRE_0_4: {$substr: ["$NOMBRE", 0, 4]}, "NOMBRE_1_3" : {$substr: ["$NOMBRE", 1, 3]}})
```

```
tutorial> db.EMPLEADOS.findOne({}, {_id: 0, Nombre: 1, NOMBRE_0_4: {$substr: ["$NOMBRE", 0, 4]}, "NOMBRE_1_3" : {$substr: ["$NOMBRE", 1, 3]}})
{ NOMBRE_0_4: 'Carl', NOMBRE_1_3: 'arl' }
tutorial>
```

Para esto utilizamos la orden **\$substr** que recibe como parámetros un arreglo con el string a cortar (en este caso utilizamos la columna \$NOMBRE), la posición donde queremos empezar a cortar y la cantidad de caracteres que queremos obtener.

REPLACE

Para este caso utilizamos la orden **\$replace** que reemplaza en una cadena un texto por otro. Le tenemos que pasar la key **input** donde se para el string a verificar, la key **find** donde le pasamos el string a buscar o ser reemplazado y la key **replacement** con el string de reemplazo.

```
db.EMPLEADOS.find({}, {_id: 00, PRODUCTO: { $replaceAll: { input: "$NOMBRE", find: "a", replacement: "__" } } })
```

```
tutorial> db.EMPLEADOS.find({}, {_id: 00, PRODUCTO: { $replaceAll: { input: "$NOMBRE", find: "a", replacement: "__" } } })
[
  { PRODUCTO: 'C__rlos' },
  { PRODUCTO: 'Elen__' },
  { PRODUCTO: 'José' },
  { PRODUCTO: 'M__rg__rit__' }
]
tutorial>
```

IF

Es el condicional simple se implementa con la orden **cond** que recibe un objeto

del con la key **if** que contiene la comparación, la key **then** con el valor del entonces o verdadero y la key **else** con el valor del sino entonces o falso de la comparación.

```
db.PERSONAS.find({}, {_id: 0, NOMBRE: 1, RUBIA: 1, RUBIA_IF: {$cond: {if: { $eq: ["$RUBIA", "S"] }, then: "Sí", else: "No"}}})
```

```
tutorial> db.PERSONAS.find({}, {_id: 0, NOMBRE: 1, RUBIA: 1, RUBIA_IF: {$cond: {if: { $eq: ["$RUBIA", "S"] }, then: "Sí", else: "No"}}})
[
  { NOMBRE: 'Manuel', RUBIA: 'S', RUBIA_IF: 'Sí' },
  { NOMBRE: 'Maria', RUBIA: 'N', RUBIA_IF: 'No' },
  { NOMBRE: 'Carmen', RUBIA: 'S', RUBIA_IF: 'Sí' },
  { NOMBRE: 'José', RUBIA: 'S', RUBIA_IF: 'Sí' },
  { NOMBRE: 'Pedro', RUBIA: 'N', RUBIA_IF: 'No' }
]
tutorial>
```

ROUND

Para este caso vamos a utilizar la orden **\$round** que recibe un array con el valor y la cantidad de decimales a redondear.

```
db.EMPLEADOS.find({}, {_id: 0, RETENCION: {$multiply: ["$SALARIO", 0.035]} ,
RETENCION_ROUND: { $round: [{ $multiply: ["$SALARIO", 0.035]}, 2] }})
```

```
tutorial> db.EMPLEADOS.find({}, {_id: 0, RETENCION: {$multiply: ["$SALARIO", 0.035]} , RETENCION_ROUND: { $round: [{ $multiply: ["$SALARIO", 0.035]}, 2] }})
[
  { RETENCION: 52.50000000000001, RETENCION_ROUND: 52.5 },
  { RETENCION: 45.50000000000001, RETENCION_ROUND: 45.5 },
  { RETENCION: 49.00000000000001, RETENCION_ROUND: 49 },
  { RETENCION: 46.39250000000005, RETENCION_ROUND: 46.39 }
]
tutorial>
```

TRUNCATE

Para este caso vamos a utilizar la orden **\$trunc** que recibe un array con el valor y la cantidad de decimales a recortar.

```
db.EMPLEADOS.find({}, {_id: 0, RETENCION: {$multiply: ["$SALARIO", 0.035]} ,
RETENCION_ROUND: { $trunc: [{ $multiply: ["$SALARIO", 0.035]}, 2] }})
```

```
tutorial> db.EMPLEADOS.find({}, {_id: 0, RETENCION: {$multiply: ["$SALARIO", 0.035]} , RETENCION_ROUND: { $trunc: [{ $multiply: ["$SALARIO", 0.035]}, 2] }})
[
  { RETENCION: 52.50000000000001, RETENCION_ROUND: 52.5 },
  { RETENCION: 45.50000000000001, RETENCION_ROUND: 45.5 },
  { RETENCION: 49.00000000000001, RETENCION_ROUND: 49 },
  { RETENCION: 46.39250000000005, RETENCION_ROUND: 46.39 }
]
tutorial>
```

Relación entre tablas

Como mencionamos anteriormente, este tipo de base de datos no es ideal para trabajar de esta forma a pesar de que algunos motores lo soporten.

Vamos a ver la colección de cursos y profesores que en las bases de datos relacionadas tienen una

relación.

```
db.PROFESORES.find()
```

```
tutorial> db.PROFESORES.find()
[
  {
    _id: ObjectId("6462b65293abdc6a0add2582"),
    ID: 1,
    NOMBRE: 'Federico',
    APELLIDOS: 'Gasco Daza',
    F_NACIMIENTO: '1975-04-23'
  },
  {
    _id: ObjectId("6462b65a93abdc6a0add2584"),
    ID: 2,
    NOMBRE: 'Ana',
    APELLIDOS: 'Saura Trenzo',
    F_NACIMIENTO: '1969-08-02'
  },
  {
    _id: ObjectId("6462b66293abdc6a0add2586"),
    ID: 3,
    NOMBRE: 'Rosa',
    APELLIDOS: 'Honrosa Pérez',
    F_NACIMIENTO: '1980-09-05'
  }
]
```

```
db.CURSOS.find()
```

```
tutorial> db.CURSOS.find()
[
  {
    _id: ObjectId("6462b7d893abdc6a0add25ac"),
    ID: 1,
    TITULO: 'Programación PHP',
    ID_PROFE: 3
  },
  {
    _id: ObjectId("6462b7de93abdc6a0add25ae"),
    ID: 2,
    TITULO: 'Modelos abstracto de datos',
    ID_PROFE: 3
  },
  {
    _id: ObjectId("6462b7e993abdc6a0add25b0"),
    ID: 3,
    TITULO: 'SQL desde cero',
    ID_PROFE: 1
  },
  {
    _id: ObjectId("6462b7ed93abdc6a0add25b2"),
    ID: 4,
    TITULO: 'Dibujo técnico',
    ID_PROFE: 2
  },
  {
    _id: ObjectId("6462b7fd93abdc6a0add25b4"),
    ID: 5,
    TITULO: 'SQL avanzado',
    ID_PROFE: null
  }
]
tutorial>
```

En caso de querer representar la misma relación, una opción sería

```
db.CURSOS_SIN_RELACION.find()
```

```
tutorial> db.CURSOS_SIN_RELACION.find()
[
  {
    _id: ObjectId("6462b7d893abdc6a0add25ac"),
    ID: 1,
    TITULO: 'Programación PHP',
    PROFESOR: {
      NOMBRE: 'Rosa',
      APELLIDOS: 'Honrosa Pérez',
      F_NACIMIENTO: '1988-09-05'
    }
  },
  {
    _id: ObjectId("6462b7de93abdc6a0add25ae"),
    ID: 2,
    TITULO: 'Modelos abstracto de datos',
    PROFESOR: {
      NOMBRE: 'Rosa',
      APELLIDOS: 'Honrosa Pérez',
      F_NACIMIENTO: '1988-09-05'
    }
  },
  {
    _id: ObjectId("6462b7e993abdc6a0add25b0"),
    ID: 3,
    TITULO: 'SQL desde cero',
    PROFESOR: {
      NOMBRE: 'Ana',
      APELLIDOS: 'Saura Trenzo',
      F_NACIMIENTO: '1989-08-02'
    }
  },
  {
    _id: ObjectId("6462b7ed93abdc6a0add25b2"),
    ID: 4,
    TITULO: 'Dibujo técnico',
    PROFESOR: {
      NOMBRE: 'Ana',
      APELLIDOS: 'Saura Trenzo',
      F_NACIMIENTO: '1989-08-02'
    }
  },
  {
    _id: ObjectId("6462b7fd93abdc6a0add25b4"),
    ID: 5,
    TITULO: 'SQL avanzado',
    PROFESOR: null
  }
]
tutorial>
```

Notese que la key **ID_PROFE** fue reemplazada por **PROFESOR** y dentro de está última tenemos un objeto con los datos del profesor. Los datos del profesor se repiten cada vez que un profesor está a cargo de un curso.