

# Bases de datos 2

Clase 8

Kevin Morán

[bases2.fiusac@gmail.com](mailto:bases2.fiusac@gmail.com)

# Document Based

## INTRODUCCIÓN A MONGODB

# Document Based

- Las bases de datos almacenan y recuperan documentos que pueden ser XML, JSON, BSON, etc.
- Los documentos almacenados son similares unos con otros pero no necesariamente con la misma estructura.

# MongoDB

- Su nombre surge de la palabra en inglés “hum**mongous**” (que significa enorme).
- MongoDB guarda estructuras de datos en documentos tipo JSON (JavaScript Object Notation) con un esquema dinámico.
- Internamente MongoDB almacena los datos en formato BSON (Binary JavaScript Object Notation).
- BSON está diseñado para tener un almacenamiento y velocidad más eficiente.



# El Origen



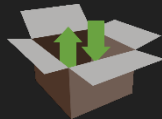
Bases de Datos  
Documentales

2007

La empresa 10gen lo desarrolla cuando estaba desarrollando una Plataforma cómo servicio (PaaS - Platform as a Service). Similar a Google App Engine.



Bases de Datos de  
Propósitos Generales



Bases de Datos  
De Código Abierto

2009

En este año MongoDB es lanzado como Producto. Es publicado bajo licencia de código abierto AGPL.

2011

Se lanza la versión 1.4 considerada como una Base de Datos lista para producción.

2016

MongoDB está por la versión 3.2.8 y es la Base de Datos NoSQL con mayor popularidad.

2021

Actualmente va por la versión 4.4.6



# Terminología RDBMS vs. Document Based (MongoDB)

RDBMS	MongoDB
Database instance	MongoDB instance
Database / Schema	Database
Table	Collection
Row	Document
Rowid	_id
Join	Dbref

# Modelado de Relaciones entre Documentos

## Relaciones Uno a Uno con documentos embebidos

### Modelo Normalizado

Si la dirección es un dato frecuentemente consultado junto con el Nombre de la persona, la mejor opción será embeber la dirección en los datos de la persona.

```
Colección Personas
{ _id: "u0001",
  nombre: "Juan Martín Hernandez" }
```

```
Colección Direcciones
{ persona_id: "u0001",
  calle: "Malabia 2277",
  ciudad: "CABA",
  provincia: "CABA",
  codPostal: "1425" }
```



```
Colección Personas
{ _id: "u0001",
  nombre: "Juan Martín Hernandez"

  direccion: { calle: "Malabia 2277",
                ciudad: "CABA",
                provincia: "CABA",
                codPostal: "1425" }
}
```

Con una sola consulta podríamos recuperar toda la información de una persona.

# Modelado de Relaciones entre Documentos

## Relaciones Uno a Muchos Con Documentos Embebidos

Si las direcciones son un dato frecuentemente consultado junto con el Nombre de la persona, la mejor opción será embeber las direcciones en los datos de la persona.

### Colección Personas

```
{ _id: "u0001",  
  nombre: "Juan Martín Hernandez" }
```

### Colección Direcciones

```
{ persona_id: "u0001",  
  calle: "Malabia 2277",  
  ciudad: "CABA",  
  provincia: "CABA",  
  codPostal: "1425" }  
  
  persona_id: "u0001",  
  calle: "Av. Santa Fe 3455",  
  ciudad: "Mar del Plata",  
  provincia: "Buenos Aires",  
  codPostal: "7600" }
```

### Colección Personas

```
{ _id: "u0001",  
  nombre: "Juan Martín Hernandez"  
  
  direcciones: [{calle: "Malabia 2277",  
                  ciudad: "CABA",  
                  provincia: "CABA",  
                  codPostal: "1425" },  
  
                {calle: "Av. Santa Fe 3455",  
                  ciudad: "Mar del Plata",  
                  provincia: "Buenos Aires",  
                  codPostal: "7600" }  
              ]  
}
```



Con una sola consulta podríamos recuperar toda la información de una persona.



# Modelado de Relaciones entre Documentos

## Relaciones Uno a Muchos Con Documentos Referenciados

### Colección libros

```
{titulo: "MongoDB: The Definitive Guide"  
  autor: [ "K. Chodorow", "M. Dirolf" ],  
  fechaPublicacion: ISODate("2010-09-24"),  
  paginas: 216,  
  lenguaje: "Ingles",  
  editor: { nombre: "O'Reilly Media",  
            anioFundacion: 1980,  
            USASState: "CA" } }
```

```
{titulo: "50 Tips and Tricks for MongoDB...",  
  autor: "K. Chodorow",  
  fechaPublicacion: ISODate("2011-05-06"),  
  paginas: 68,  
  lenguaje: "Ingles",  
  editor: { nombre: "O'Reilly Media",  
            anioFundacion: 1980,  
            USASState: "CA" } }
```



### Colección Editores

```
{ nombre: "O'Reilly Media",  
  anioFundacion: 1980,  
  USASState: "CA",  
  libros: [987654321, 1234567890] }
```

### Colección Libros

```
{_id: 987654321  
  titulo: "MongoDB: The Definitive Guide"  
  autor: [ "K. Chodorow", "M. Dirolf" ],  
  fechaPublicacion: ISODate("2010-09-24"),  
  paginas: 216,  
  lenguaje: "Ingles"}  
{_id: 1234567890  
  titulo: "50 Tips and Tricks for MongoDB...",  
  autor: "K. Chodorow",  
  fechaPublicacion: ISODate("2011-05-06"),  
  paginas: 68,  
  lenguaje: "Ingles"}
```

Cuando usamos referencias, el crecimiento de las relaciones determinan donde conviene almacenar la referencia. Por ej. Si el nro. de libros por editor es chico y no crecerá mucho, este modelo podría ser conveniente.

# Modelado de Relaciones entre Documentos

## Relaciones Uno a Muchos Con Documentos Referenciados

### Colección libros

```
{titulo: "MongoDB: The Definitive Guide",
 autor:[ "K. Chodorow", "M. Dirolf" ],
 fechaPublicacion: ISODate("2010-09-24"),
 paginas: 216,
 lenguaje: "Ingles",
 editor: { nombre: "O'Reilly Media",
          anioFundacion: 1980,
          USASState: "CA" } }
```

```
{titulo: "50 Tips and Tricks for MongoDB...",
 autor: "K. Chodorow",
 fechaPublicacion: ISODate("2011-05-06"),
 paginas: 68,
 lenguaje: "Ingles",
 editor: { nombre: "O'Reilly Media",
          anioFundacion: 1980,
          USASState: "CA" } }
```



### Colección Editores

```
{ _id: "oreilly",
  nombre: "O'Reilly Media",
  anioFundacion: 1980,
  USASState: "CA",
}
```

### Colección Libros

```
{_id: 987654321
 titulo: "MongoDB: The Definitive Guide"
 autor:[ "K. Chodorow", "M. Dirolf" ],
 fechaPublicacion: ISODate("2010-09-24"),
 paginas: 216,
 lenguaje: "Ingles",
 idEditor: "oreilly"}
```

```
{_id: 1234567890
 titulo: "50 Tips and Tricks for MongoDB...",
 autor: "K. Chodorow",
 fechaPublicacion: ISODate("2011-05-06"),
 paginas: 68,
 lenguaje: "Ingles",
 idEditor: "oreilly"}
```

En cambio si queremos evitar Arreglos mutables y crecientes podemos implementar una referencia al editor dentro de cada libro.

# En qué casos usarlas ?

## Logging de Eventos

- las bases de datos basadas en documentos puede loguear cualquier clase de eventos y almacenarlos con sus diferentes estructuras.
- Pueden funcionar como un repositorio central de logueo de eventos.

## CMS, blogging

- su falta de estructura predefinida hace que funcionen bien para este tipo de aplicaciones.

## Web-analytics / Real-Time analytics

- Almacenar cantidad de vistas a una página o visitantes únicos.

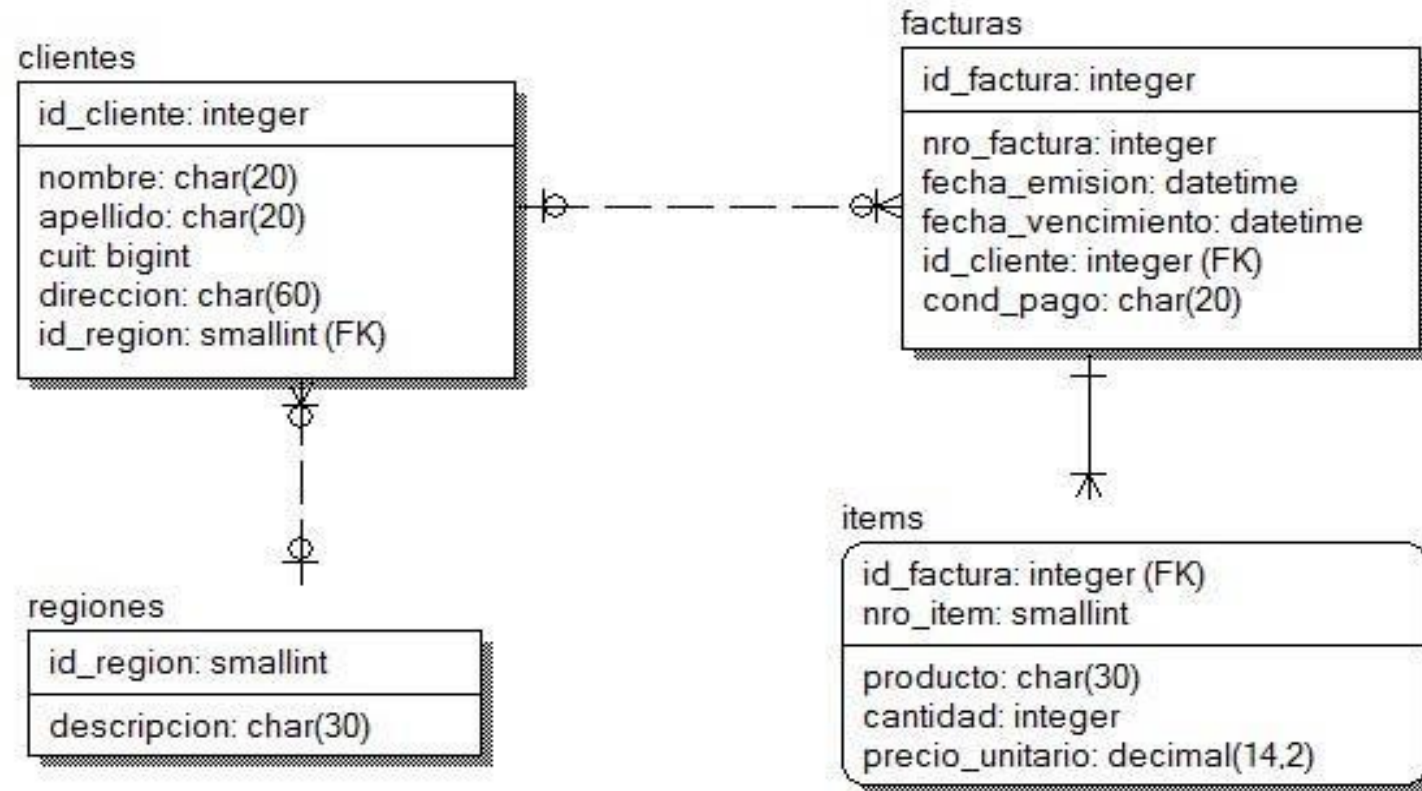
## Commerce

- A menudo requieren tener esquemas flexibles para los productos y órdenes

# UTILIZANDO MONGODB

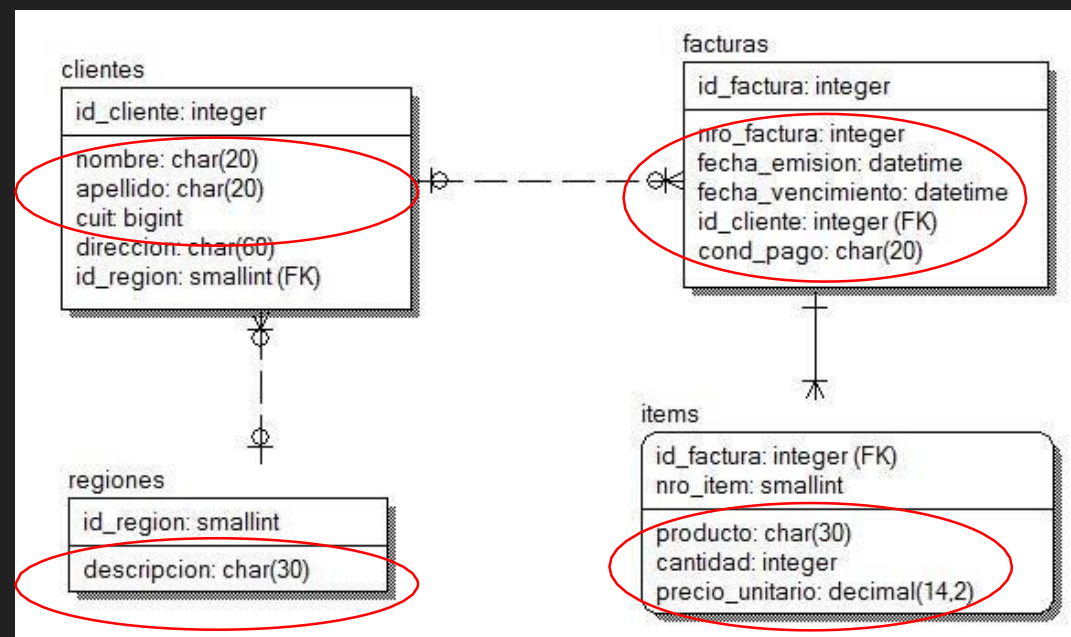


# Caso Práctico



# Caso Práctico

Armaremos un modelo que contenga la información de las facturas y todos sus ítems, detallando el nombre, apellido, NIT y región del cliente al que se le emitió la factura, para poder realizar consultas desde un portal de facturas de la forma más rápida posible.





# Caso Práctico

## Insertar

```
db.facturas.insert({"_id":
```


```
  ObjectId("55e4a6fabfc68c676a041063"), "cliente": {"apellido": "Malinez", "cuit": 2.740488484e+09, "nombre": "Marina", "region": "CENTRO"}, "condPago": "CONTADO", "fechaEmision": ISODate("2014-02-20T00:00:00.000Z"), "fechaVencimiento": ISODate("2014-02-20T00:00:00.000Z"), "item": [{"cantidad": 11.0, "precio": 18.0, "producto": "CORREA 12mm"}, {"cantidad": 1.0, "precio": 490.0, "producto": "TALADRO 12mm"}], "nroFactura": 1000.0})
```

```
db.facturas.insert({"_id":
```

```
  ObjectId("55e4a6fbbfc68c676a041064"), "cliente": {"apellido": "Zavasi", "cuit": 2.038373771e+09, "nombre": "Martin", "region": "CABA"}, "condPago": "30 Ds FF", "fechaEmision": ISODate("2014-02-20T00:00:00.000Z"), "fechaVencimiento": ISODate("2014-03-22T00:00:00.000Z"), "item": [{"cantidad": 2.0, "precio": 134.0, "producto": "CORREA 10mm"}], "nroFactura": 1001.0})
```

```
db.facturas.insert({"_id":
```

```
  ObjectId("55e4a6fbbfc68c676a041065"), "cliente": {"apellido": "Zavasi", "cuit": 2.038373771e+09, "nombre": "Martin", "region": "CABA"}, "condPago": "CONTADO", "fechaEmision": ISODate("2014-02-20T00:00:00.000Z"), "fechaVencimiento": ISODate("2014-02-20T00:00:00.000Z"), "item": [{"cantidad": 6.0, "precio": 60.0, "producto": "TUERCA 2mm"}, {"cantidad": 12.0, "precio": 134.0, "producto": "CORREA 10mm"}], "nroFactura": 1002.0})
```



# UTILIZACIÓN DE MONGODB DESDE LA LÍNEA DE COMANDOS

# Operaciones sobre una colección

## **Ver todos los comandos:**

```
db.facturas.help()
```

Ejemplos:

## **Ver todos los documentos:**

```
db.facturas.find()
```

## **Cantidad de documentos en la colección:**

```
db.facturas.count()
```

## **Espacio ocupado por los documentos de la colección:**

```
db.facturas.dataSize()
```

# Operaciones sobre una colección

## **Inicio de servidor mongo por consola:**

C:>mongod

## **Inicio de cliente mongo por consola:**

C:>mongo

## **Ver bases de datos**

show dbs

## **Ver colecciones**

show collections

# Operaciones sobre una colección

## **Buscar documento para cliente con determinado apellido**

```
db.facturas.find({"cliente.apellido":"Malinez"})
```

## **Consultar los primeros dos documentos**

```
db.facturas.find().limit(2)
```

## **Consultar documentos saltando los primeros dos documentos**

```
db.facturas.find().skip(2)
```

## **Visualización mejorada**

```
db.facturas.find().pretty()
```

# Operaciones sobre una colección

**Consultar dos documentos, saltando los dos primeros documentos de una colección, mostrándolos en un modo mejorado.**

```
db.facturas.find().limit(2).skip(2).pretty()
```

**Consultar documentos sólo mostrando algunos datos**

```
db.facturas.find({"cliente.apellido":"Malinez"}, {"cliente.cuit":1, "cliente.region":1})
```

**Buscar documento para cliente con dos criterios**

Para Zavasi teníamos dos documentos: `db.facturas.find({"cliente.apellido":"Zavasi"}).pretty()`

Agregamos un criterio: `db.facturas.find({"cliente.apellido":"Zavasi", "nroFactura":1001.0}).pretty()`

**Ordenamiento en forma ascendente**

```
db.facturas.find().sort({nroFactura:1})
```

**Ordenamiento en forma descendente**

```
db.facturas.find().sort({nroFactura:-1})
```



# Consultando una Colección – Criterios de Selección

## Operadores \$

\$gt  
\$gte  
\$lt  
\$lte  
\$not  
\$or  
\$in  
\$nin  
\$exist  
\$regex

Busca la cantidad de facturas cuyo Nro. de Factura sea mayor que 1465

```
db.facturas.find( { nroFactura : { $gt: 1465 } } ).count()
```

```
>  
> db.facturas.find( { nroFactura : { $gt: 1465 } } ).count()  
30512  
>
```

Busca las facturas cuya fecha de emisión sea mayor o igual al 24/02/2014.

```
db.facturas.find( { fechaEmision: { $gte: ISODate("2014-02-24T00:00:00Z") } } )
```

```
> db.facturas.find( { fechaEmision: { $gte: ISODate("2014-02-24T00:00:00Z") } } )  
< { "_id" : ObjectId("53685dbb2baf7b93f61df567"), "nroFactura" : 1450, "fechaEmisi  
on" : ISODate("2014-02-24T00:00:00Z"), "fechaVencimiento" : ISODate("2014-02-24T  
00:00:00Z"), "condPago" : "CONTADO", "cliente" : { "nombre" : "Juan Manuel", "ap  
ellido" : "Manoni", "cuit" : 2029889382, "region" : "NEA" }, "item" : [   
  { "producto" : "TUERCA 2mm", "cantidad" : 2, "precio" : 60 },  
  { "producto" : "TALADRO 12mm", "cantidad" : 1, "precio" : 490 },  
  { "producto" : "TUERCA 5mm", "cantidad" : 15, "precio" : 90 } ] }  
< { "_id" : ObjectId("53685dbb2baf7b93f61df568"), "nroFactura" : 1451, "fechaEmisi  
on" : ISODate("2014-02-24T00:00:00Z"), "fechaVencimiento" : ISODate("2014-03-26T  
00:00:00Z"), "condPago" : "30 Ds FF", "cliente" : { "nombre" : "Soledad", "apell  
ido" : "Lavagno", "cuit" : 2729887543, "region" : "NOA" }, "item" : [ { "produ
```

# Insertando un Documento

## El método insert tiene la siguiente sintaxis:

Evalúa si existe un próximo documento. Devuelve True o False.

```
db.collection.insert  
( <document or array of documents>,  
  { writeConcern: <document>,  
    ordered: <boolean> } )
```

**writeConcern**

Es opcional, lo veremos en la parte de consistencia.

**Ordered**


lo vemos en un par de slides

## Ejemplo, inserción de un documento sin \_id:

```
db.facturas.insert({nroFactura:30003,codPago:"CONTADO"})
```

**\_id:** Document Id único autogenerated

```
> db.facturas.insert(<nroFactura:30003,codPago:"CONTADO">)  
WriteResult({ "nInserted" : 1 })  
>  
>  
> db.facturas.find(<nroFactura:30003>)  
{ "_id" : ObjectId("5459a129cc19250561ad5f82"), "nroFactura" : 30003, "codPago"  
: "CONTADO" }
```



# Insertando un Documento

**Ejemplo, inserción de un documento con `_id`:**

```
db.facturas.insert({_id:23094776, nroFactura:30004,codPago:"CONTADO"})
```

```
> db.facturas.insert(<{_id:23094776,nroFactura:30004,codPago:"CONTADO"}>
WriteResult(< { "nInserted" : 1 } >)
>
>
> db.facturas.find(<nroFactura:30004>)
{ "_id" : 23094776, "nroFactura" : 30004, "codPago" : "CONTADO" }
```

Al crear una colección, el motor de BD crea un índice único sobre el atributo `_id`.

```
> db.facturas.insert(<{_id:23094776,nroFactura:30004,codPago:"30dsFF"}>
WriteResult(<
  "nInserted" : 0,
  "writeError" : {
    "code" : 11000,
    "errmsg" : "insertDocument :: caused by :: 11000 E11000 duplicat
e key error index: finanzas.facturas.$_id_ dup key: { : 23094776.0 }"
  }
>>
```

# Borrando Documentos

## Operación Remove

### Sintaxis

```
db.<collection_name>.remove({criterio_de Eliminación})
```

Esta operación eliminará los documentos que cumplan con el criterio definido.

**Warning: Remove es una operación de tipo multi-documento!!**

*Recomendación: Es conveniente antes de borrar hacer un find o un count para asegurarse lo que quiero borrar.*

### Ejemplo 1 – Borrado de TODOS LOS DOCUMENTOS de una colección

```
db.accesos.remove({})
```

Elimina **TODOS LOS ELEMENTOS** de una colección.

```
> db.accesos.remove({})
WriteResult(< "nRemoved" : 3 >)
>
> db.accesos.find()
```

# Borrando Documentos

## Ejemplo 2 – Remove por clave primaria

```
db.updtst.remove({_id:100})
```

Elimina el documento cuyo `_id` sea 100 de la colección **updtst**.

```
> db.updtst.remove({_id:100})
WriteResult(< "nRemoved" : 1 >)
>
> db.updtst.find()
{ "_id" : 300, "items" : [ 88, 99, 97 ] }
{ "_id" : 200 }
```

## Ejemplo 3 – Remove por un criterio con múltiples documentos que aplican

```
db.updtst.remove({items:88})
```

```
> db.updtst.find()
{ "_id" : 300, "items" : [ 88, 99, 97 ] }
{ "_id" : 500, "items" : 88 }
{ "_id" : 200 }
{ "_id" : 400, "items" : [ 77, 88 ] }
> db.updtst.remove({items:88})
WriteResult({ "nRemoved" : 3 })
> db.updtst.find()
{ "_id" : 200 }
```

# Modificando Documentos

Permite modificar uno o más documentos de una colección. Por default modifica sólo un documento.

```
db.coleccion.update ( {clausula_where},  
                      {documento_o_expresión_a_modificar},  
                      { upsert, multi, writeconcern}  
                      )
```

**upsert** (true o false) Si está configurado en “True” significa que realizará un update si existe un documento que concuerda con el criterio, o un insert si no existe algún documento que concuerde con el criterio. El valor default es “false”, en este caso no realiza un insert cuando no existe documento que concuerde con el criterio.

**multi** (true o false) Es opcional. Si es configurado en true, el update realiza la actualización de multiples documentos que concuerdan con el criterio cláusula\_where. Si es configurado en false, modifica solo un documento. El valor default es false. Sólo actúa en updates parciales con operadores \$.

**writeconcern** Es opcional, lo veremos en la parte de consistencia.



# Modificando Documentos

## Update Totales/Completo

Se realiza el update del documento completo, reemplazando el mismo.

## Update Parciales

### Operadores

#### Operadores sobre cualquier atributo

\$set	Permite modificar el valor de un atributo, o agregar un nuevo atributo al documento. Permite eliminar un atributo
\$unset	de un documento.
\$inc	Incrementa o decrementa el valor de un atributo ( n ó -n)

#### Operadores sobre Arrays

\$push	Agrega un elemento a un Array o crea un Array con un elemento. Agrega un elemento
\$addToSet	al Array solo si no existe en el Array.
\$pushAll	Agrega varios elementos a un Array con los valores indicados o crea un Array con esos elementos. <i>(Operación Múltiple)</i>
\$pop	Elimina un elemento de un Array por sus extremos, permitiendo eliminar el primer elemento (-1) o el último (1). Elimina todos los elementos de un Array que contengan el valor indicado. Elimina todos los elementos de un Array que contengan alguno de los valores indicados.
\$pull	<i>(Operación Múltiple)</i>
\$pullAll	

# Modificando Documentos Completos

## Update Totales/Completos

```
db.updtst.update({x:2},{ "x" : 2, "y" : 999 })
```

Este comando reemplaza el primer documento encontrado por con valor x:2 por este otro en donde el elemento y:999, no tengo el control de cuál estoy modificando, lo correcto era modificar poniendo en el criterio el `_id`.

```
> db.updtst.update({x:2},{ "x" : 2, "y" : 999 })
> db.updtst.find()
{ "_id" : ObjectId("536a8240793253ebed598065"), "x" : 1, "y" : 999 }
{ "_id" : ObjectId("536a8245793253ebed598066"), "x" : 2, "y" : 999 }
{ "_id" : ObjectId("536a8248793253ebed598067"), "x" : 2, "y" : 100 }
{ "_id" : ObjectId("536a824b793253ebed598068"), "x" : 2, "y" : 300 }
{ "_id" : ObjectId("536a8250793253ebed598069"), "x" : 3, "y" : 100 }
{ "_id" : ObjectId("536a8254793253ebed59806a"), "x" : 3, "y" : 200 }
{ "_id" : ObjectId("536a8257793253ebed59806b"), "x" : 3, "y" : 300 }
```

# Modificando Documentos Parciales

## Update Parciales

### Ejemplo 1 – Operador \$set – Modificación de un valor de un atributo existente

Dado el siguiente documento:

```
> db.updtst.insert({_id:100,x:10,y:100})
```

```
db.updtst.update({_id:100},{ $set : {x:100}})
```

Realizará una modificación del valor de atributo x a 100

```
> db.updtst.find(< id:100>)  
{ "_id" : 100, "x" : 100, "y" : 100 }
```

# Modificando Documentos Parciales

## Update Parciales

Otro Ejemplo – Operador \$set – Opción multi – Agregar un atributo en todos los documentos

```
db.updtst.update({x:2},{ $set : {z:"NUEVO"}},{multi:true})
```

Este reemplaza en TODOS los documentos encontrados con valor x:2 agregando el atributo z:"NUEVO"

```
> db.updtst.update({x:2},{ $set : {z:"NUEVO"}},{multi:true})
> db.updtst.find({x:2})
{ "_id" : ObjectId("536a8245793253ebed598066"), "x" : 2, "y" : 999, "z" : "NUEVO"
  }
{ "_id" : ObjectId("536a8248793253ebed598067"), "x" : 2, "y" : 100, "z" : "NUEVO"
  }
{ "_id" : ObjectId("536a824b793253ebed598068"), "x" : 2, "y" : 300, "z" : "NUEVO"
  }
```