



Bases de datos Documentales

Document Oriented

Permiten la insercion, obtención y manipulación de información en formato semi estructurado llamado **Documento**.

Basicamente un documento es un conjunto de claves-valor, donde estos valores pueden contener otros documentos.



Document data model

Document Oriented

Usualmente el documento está en alguno de los siguientes formatos:

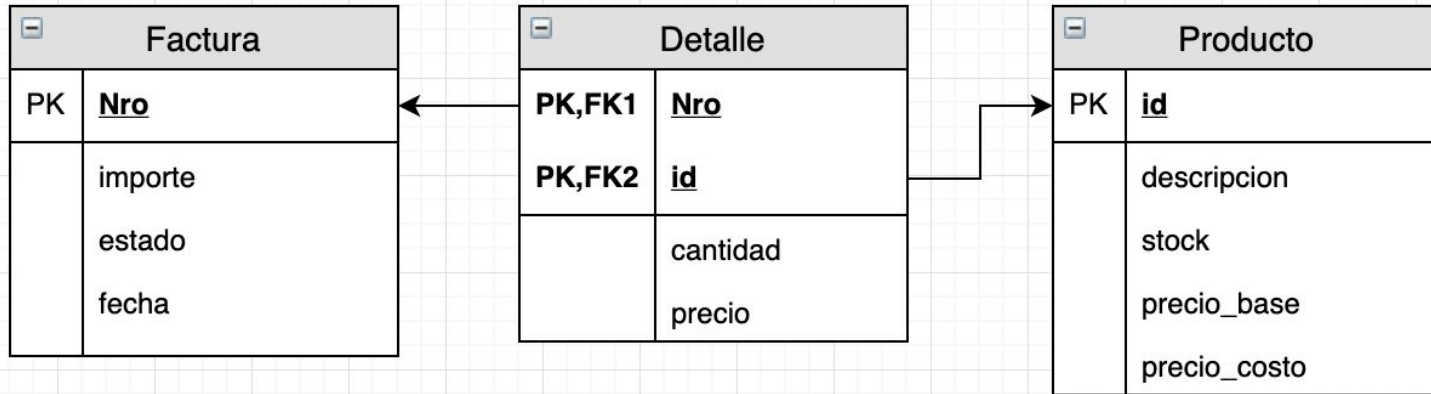
- XML
- JSON
- YAML

```
- <employees>
- <person id="1392">
  <name>John Smith</name>
  <dob>1974-07-25</dob>
  <start-date>2004-08-01</start-date>
  <salary currency="USD">35000</salary>
</person>
- <person id="1395">
  <name>Clara Tennison</name>
  <dob>1968-03-15</dob>
  <start-date>2003-05-16</start-date>
  <salary currency="USD">27000</salary>
</person>
</employees>
```

Ejemplo de XML

Document Oriented - Ejemplo factura

Esta es una posible representación del ejemplo de factura en una BD relacional



Document Oriented -- XML - JSON - YAML

```
<?xml version="1.0" encoding="UTF-8" ?>
<factura>
  <nro>123</nro>
  <importe>100</importe>
  <estado>iniciada</estado>
  <fecha>10/03/2019</fecha>
  <detalles>
    <detalle>
      <cantidad>2</cantidad>
      <precio>35.5</precio>
      <subtotal>71</subtotal>
      <producto>
        <id>456</id>
        <descripcion>algo</descripcion>
      </producto>
    </detalle>
    <detalle>
      <precio>29</precio>
      <cantidad>1</cantidad>
      <producto>algo diferente</producto>
    </detalle>
  </detalles>
</factura>
```

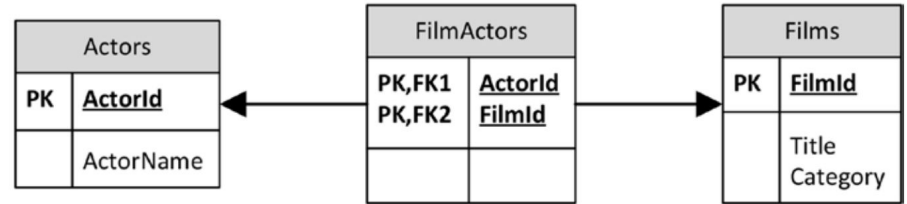
```
{
  "nro":123,
  "importe":100,
  "estado":"iniciada",
  "fecha":"10/03/2019",
  "detalles":[
    {
      "cantidad":2,
      "precio":35.5,
      "subtotal":71,
      "producto":{
        "id":456,
        "descripcion":"algo"
      }
    },
    {
      "precio":29,
      "cantidad":1,
      "producto":"algo diferente"
    }
  ]
}
```

```
---
nro: 123
importe: 100
estado: iniciada
fecha: 10/03/2019
detalles:
  - cantidad: 2
    precio: 35.5
    subtotal: 71
    producto:
      id: 456
      descripcion: algo
  - precio: 29
    cantidad: 1
    producto: algo diferente
```

Document Oriented

No tienen un esquema, esto significa que los documentos y sus 'campos' pueden variar de documento a documento tanto en tipo de dato como estructura.

Esto permite gran flexibilidad al programador, rápida respuesta a cambios y un desarrollo orientado a prototipos.



```
{ "_id": 115, "Title": "CAMPUS REMEMBER",
  "Category": "Action",
  "Actors" :
    [
      { "actorId": 8, "Name": "MATTHEW JOHANSSON" },
      { "actorId": 45, "Name": "REESE KILMER" },
      { "actorId": 168, "Name": "WILL WILSON" }
    ]
}
```

Actor document

Document Oriented

Se alienta hacer un diseño desnormalizado donde los documentos contengan otros documentos embebidos, esto es así ya que en una BD orientada a documento se intenta no realizar ningún tipo de 'Join', incrementando significativamente la velocidad de consultas y recuperación de datos.

Estas bases de datos también ofrecen potentes lenguajes de consulta y manipulación de los datos, permitiendo juntar documentos, modificarlos, procesos batch, updates masivos, etc

23 de agosto



Entregado

Llegó el 25 de agosto ⚡ FULL

Yerba Mate Playadito Suave 1000grs.
1 unidad

Mercado Libre CPG

[Enviar mensaje](#)

[Ver compra](#)

[Volver a comprar](#)



Entregado

Llegó el 25 de agosto ⚡ FULL

Sal Fina Dos Anclas Paquete X 500 Grs
1 unidad

MAYORISTANETCOM SA .-

[Enviar mensaje](#)

[Ver compra](#)

[Volver a comprar](#)

En una BD Documental, cada compra puede ser un documento con todos los datos necesarios para mostrar en esta pantalla.

Esto significa que la BD debe realizar una sola lectura (por compra).

Compra

- Producto
 - Id, nombre, cantidad
 - Vendedor { id, nombre }

En una BD Relacional se debe hacer un join de cada compra a todos sus detalles, y de cada detalle al producto y de cada producto a su vendedor.

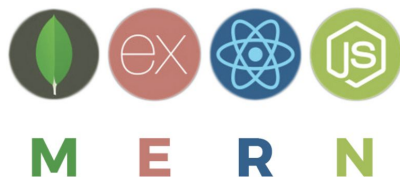
Esto significa que por cada factura deberá realizar como mínimo 3 JOINS para traer la información necesaria.

Factura JOIN Detalle JOIN producto JOIN vendedor

Document Oriented

Si se utiliza formato JSON en la Base de Datos y javascript en el servidor, el trabajo de mapeo de formatos se reduce drásticamente, resultando en una persistencia muy transparente.

Esto hace que las BD Documentales están tomando mucho auge en el desarrollo web.



MongoDB

“MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud era.”

“MongoDB is a document database, which means it stores data in **JSON-like documents**. We believe this is the most natural way to think about data, and is much more expressive and powerful than the traditional row/column model.”

MongoDb, <https://www.mongodb.com/>

- Soporta arrays, documentos embebidos o referenciados y esquemas dinámicos o estructurados
- Potente motor de búsquedas, con agregaciones, búsquedas geo-referenciales, de texto y hacer join de documentos
- Las consultas son JSON y se pueden escribir “procedimientos” en javascript. (reemplazar PLSQL)
- Soporte para transacciones ACID

Modelo de datos - JSON

JSON → JavaScript Object Notation

Es una sintaxis para guardar e intercambiar información.

Es formato de texto independiente del lenguaje que lo está utilizando.

Es una colección no ordenada de claves-valor. Soporta los siguientes tipos de datos:

- Números: positivos, negativos y con parte fraccionaria
- Cadenas(String): Representan secuencias de cero o más caracteres. Se ponen entre doble comilla.
- Booleanos: true y false
- null: Representan el valor nulo.
- Array: Representa una lista de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se define con corchetes. Ejemplo: ["juan", 123, {}]
- Objetos: Otros objetos JSON

```
{
  "nro":123,
  "importe":100,
  "estado":"iniciada",
  "fecha":"10/03/2019",
  "detalles":[
    {
      "cantidad":2,
      "precio":35.5,
      "subtotal":71,
      "producto":{
        "id":456,
        "descripcion":"algo"
      }
    },
    {
      "precio":29,
      "cantidad":1,
      "producto":"algo diferente"
    }
  ]
}
```

Modelo de datos

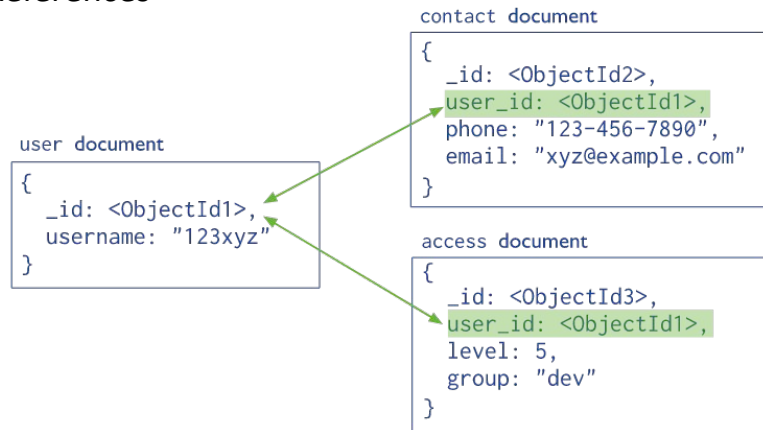
En MongoDB los documentos son estructuras de tipo JSON. Estos no tienen un esquema definido por defecto.

Es de vital importancia pensar en el uso que se dará a la información y cómo modelar sus relaciones. MongoDB permite hacerlo de dos formas.

Embedded Data



References



Embedded vs References

Las ventajas del **modelo Embedded** es que al guardar la información dentro de un mismo registro, se necesitan menos consultas y updates para realizar operaciones comunes.

Muchas veces la información tiene relaciones del tipo “contiene”, “padre-hijo” o donde el documento embebido siempre es mostrado en el contexto de su superior.

A su vez, el **modelo con referencias** ofrece un ahorro en almacenamiento al no duplicar información, y suele utilizarse en relaciones del tipo muchos a muchos o cuando la optimización de la performance no es superior al ahorro de almacenamiento.

Mongo permite usar ambos modelos al mismo tiempo y dejar la implementación del mas conveniente al desarrollador, aunque también proveen algunos ejemplos y patrones:

<https://docs.mongodb.com/manual/applications/data-models/>

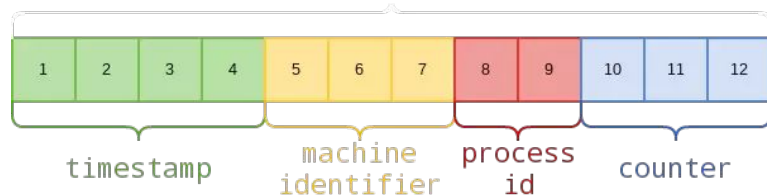
Características de MongoDB

JSON → Documento

Por defecto los documentos son flexibles, pero se puede definir “schema validation” para forzar a los documentos a tener campos definidos, tipos de datos o valores permitidos.

Los documentos se agrupan en una colección (análogos a tablas de SQL), pudiendo tener distintas estructuras.

Cada documento tiene un `_ID` que lo identifica dentro de la colección en la que se encuentra, el `_ID` puede ser definido por el usuario, de no serlo MongoDB le asignará uno. →→



Las colecciones se pueden crear de antemano o al guardar un documento en una colección que no existe, MongoDB la crea automáticamente.

MongoDB provee distintos mecanismos de seguridad como autenticación, autorización (role-based), encriptación de comunicación y/o datos, auditoría, network exposure, entre otras. En una instalación local por defecto todas están desactivadas.

Preparando MongoDB

Para instalar Mongo se recomienda seguir la guía oficial la cual provee los pasos para cada sistema operativo:

<https://docs.mongodb.com/manual/installation/>

Usaremos la versión COMUNITARIA.

También se puede usar Docker para descargar una imagen desde https://hub.docker.com/_/mongo/

Una vez instalado la BD, nos conectamos mediante el cliente de línea de comandos(mongo shell) o una interfaz gráfica como: <https://www.mongodb.com/download-center/compass>

Una vez conectados podemos realizar un par de ABMs y queries.
Los ejemplos son usando mongo shell.

(para jugar con una terminal online se puede usar <https://www.mplay.run/mongodb-online-terminal>,
<https://www.jdoodle.com/online-mongodb-terminal/>)

Interactuando con MongoDB

Nos conectamos con mongo:

Para insertar un usuario usamos el comando `insertOne(JSON)`.

En este comando estamos insertando un documento, a la coleccion **users**.

Para recuperar un dato ejecutamos el comando `find()` a la coleccion **users**.

Y si queremos filtrar?

```
lucas@debian-omen:~$ mongo
MongoDB shell version v4.2.7
connecting to: mongodb://127.0.0.1:27017/
Implicit session: session { "id" : UUID("15e120d3-8254-4682-81c2-a800fffb7942") }
MongoDB server version: 4.2.3
Welcome to the MongoDB shell.

> db.users.insertOne({name:"sue",age:26,status:"pending"})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5ecc2f02de362a00ef145ef6")
}

> db.users.find()
{
  "_id" : ObjectId("5ecc2f02de362a00ef145ef6"),
  "name" : "sue",
  "age" : 26,
  "status" : "pending"
}
```


Interactuando con MongoDB - find()

El método find() puede recibir como parámetros un JSON para filtrar, osea, compara cada documento de la colección contra el JSON y retornara aquellos que matcheen.

Invocando find({}) retorna todos los documentos de la colección. (como un select * from coleccion)

Invocando find({name: "sue"}) retornar los documentos con campo **name** que tenga como valor **sue**

Osea, filtrando por igualdad se sigue la regla: { <field1>: <value1>, ... }

En cambio si quieres usar otro criterio, deberemos usar la expresión con operadores, que sigue la regla:
{ <field1>: { <operator1>: <value1> }, ... }

Invocando find({age: {"\$gt": 20}}) retorna los documentos con el campo edad **MAYOR (greater)** a 20.

Para una lista completa de todos los operadores entrar en

<https://docs.mongodb.com/manual/reference/operator/query/>

Interactuando con MongoDB

Usamos otra colección para hacer unas queries más complejas:

```
db.inventory.insertMany([
  { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
  { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
  { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
  { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
  { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

Interactuando con MongoDB

Para hacer un AND:

```
> db.inventory.find( { status: "A", qty: { $lt: 30 } } )  
{ "_id" : ObjectId("5ee4e20237e0f928dc0d391d"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" :  
21, "uom" : "cm" }, "status" : "A" }
```

Para hacer un OR:

```
> db.inventory.find( { $or: [ { status: "D" }, { qty: { $lt: 30 } } ] } )  
{ "_id" : ObjectId("5ee4e20237e0f928dc0d391d"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" :  
21, "uom" : "cm" }, "status" : "A" }  
{ "_id" : ObjectId("5ee4e20237e0f928dc0d391f"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" :  
11, "uom" : "in" }, "status" : "D" }  
{ "_id" : ObjectId("5ee4e20237e0f928dc0d3920"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "w" :  
30, "uom" : "cm" }, "status" : "D" }
```

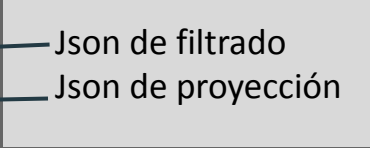
Interactuando con MongoDB

Juntando AND y OR

```
>db.inventory.find( {  
  status: "A",  
  $or: [ { qty: { $lt: 10 } }, { qty: { $gte: 100 } } ]  
})
```

Proyecciones (solo mostrar ciertos campos)

```
> db.inventory.find(  
  { status: "A" },  
  { item: 1, status: 1 }  
)  
{ "_id" : ObjectId("5ee4e20237e0f928dc0d391d"), "item" : "journal", "status" : "A" }  
{ "_id" : ObjectId("5ee4e20237e0f928dc0d391e"), "item" : "notebook", "status" : "A" }  
{ "_id" : ObjectId("5ee4e20237e0f928dc0d3921"), "item" : "postcard", "status" : "A" }
```



Interactuando con MongoDB

Query por objetos embebidos:

```
> db.inventory.find( { "size.uom" : "cm" } )
{ "_id" : ObjectId("5ee4e20237e0f928dc0d391d"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
```

Para hacer queries con objetos embebidos en un array, supongamos el siguiente caso:

```
db.inventory.insertMany( [
  { item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },
  { item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },
]
```

```
db.inventory.find( { "instock": { warehouse: "A", qty: 5 } } )
```

```
db.inventory.find( { 'instock.qty': { $lte: 20 } } )
```

← Comparación exacta de Json

← Comparación de un solo atributo

Interactuando con MongoDB

Deletes → Para borrar se usa el mismo mecanismo de filtrado que para el find()

```
>db.inventory.deleteOne( {item: "journal"} )  
>db.inventory.deleteMany( {} )
```

Updates → Para actualizar documentos se pueden usar 2 comandos: update y replace

```
db.inventory.updateOne(  
  { item: "paper" }, ← Jsn de filtrado  
  {  
    $set: { "status": "D" } ← Campos a actualizar  
  }  
)  
  
db.inventory.replaceOne(  
  { item: "paper" }, ← Jsn de filtrado  
  { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] } ← Nuevo JSON  
)
```