

# MongoDB - Bases de datos documentales

---

## Bases de datos II

Edgar Talavera Muñoz ([e.talavera@upm.es](mailto:e.talavera@upm.es))

Departamento de Sistemas Informáticos

Escuela Técnica superior de Ingeniería de Sistemas Informáticos

License CC BY-NC-SA 4.0

# **Introducción**

# Sistemas de almacenamiento

---

## Datos **estructurados**

- Hojas de calculo
- Bases de datos relacionales

## Datos **semi-estructurados** o **no estructurados**

- Se necesita un rediseño del sistema de almacenamiento

# Características de MongoDB

---

Es un motor open-source de **base de datos documental** de código abierto

- MongoDB ("humongous"), disponible en <http://www.mongodb.org>
- Líder de las bases de datos **NoSQL**

Licenciado bajo licencias libres

- Primero [GNU AGPL v3.0](#), ahora [Server Side Public License \(SSPL\)](#)
- Existen disponibles licencias comerciales para su uso en aplicaciones cerradas

Usa **UTF-8** como codificación (por defecto)

# JSON: JavaScript Object Notation

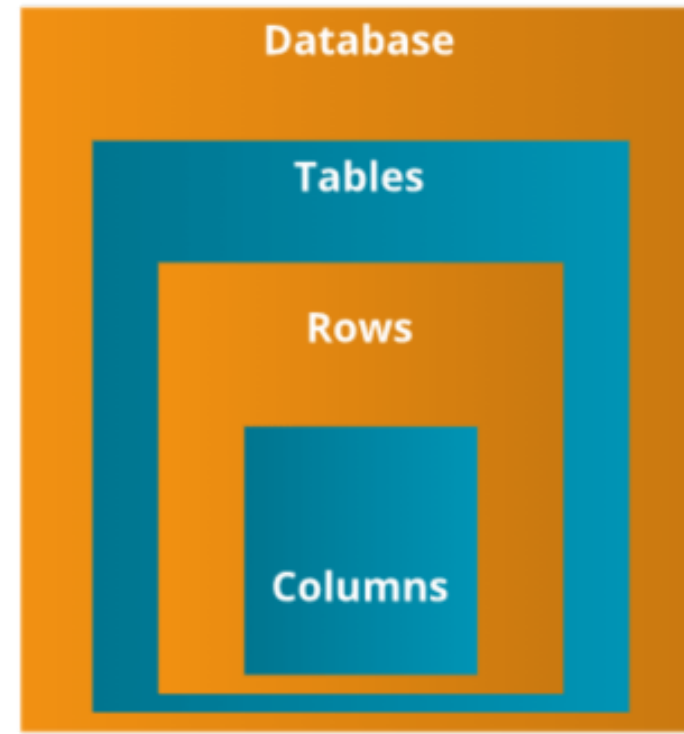
---

La información en MongoDB utiliza un formato basado en **JSON** para su sintaxis:

```
{
  "clientes": [
    {
      "apellido": "Alonso",
      "gasto": 100,
      "es_habitual": true,
      "productos": ["P001", "P032", "P099"]
    },
    ...
  ]
}
```

# MongoDB vs SQL

---



# Documentos

---

MongoDB almacena la información en forma de **documentos**

- ... que son pares clave-valor en formato **JSON**

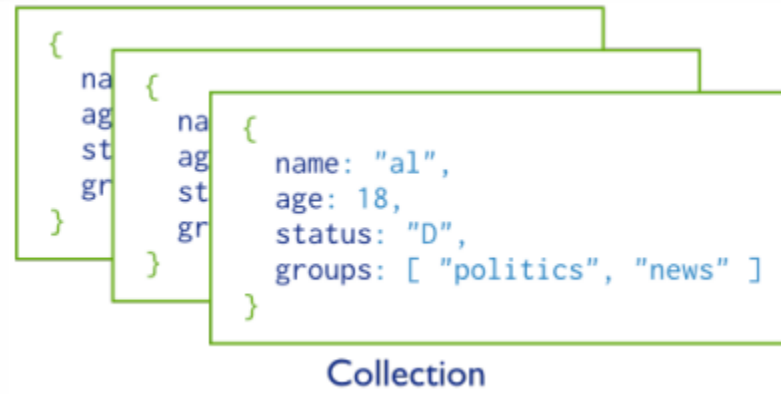
```
{  
  "clave": "valor",  
  "nombre": "Edgar",  
  "edad": 40,  
  "hobbies": ["Frodo", "Sam", "Miss Pelotas"]  
}
```

# Colecciones

---

MongoDB almacena todos los documentos en **colecciones**

- Una colección es un **grupo de documentos relacionados** semánticamente

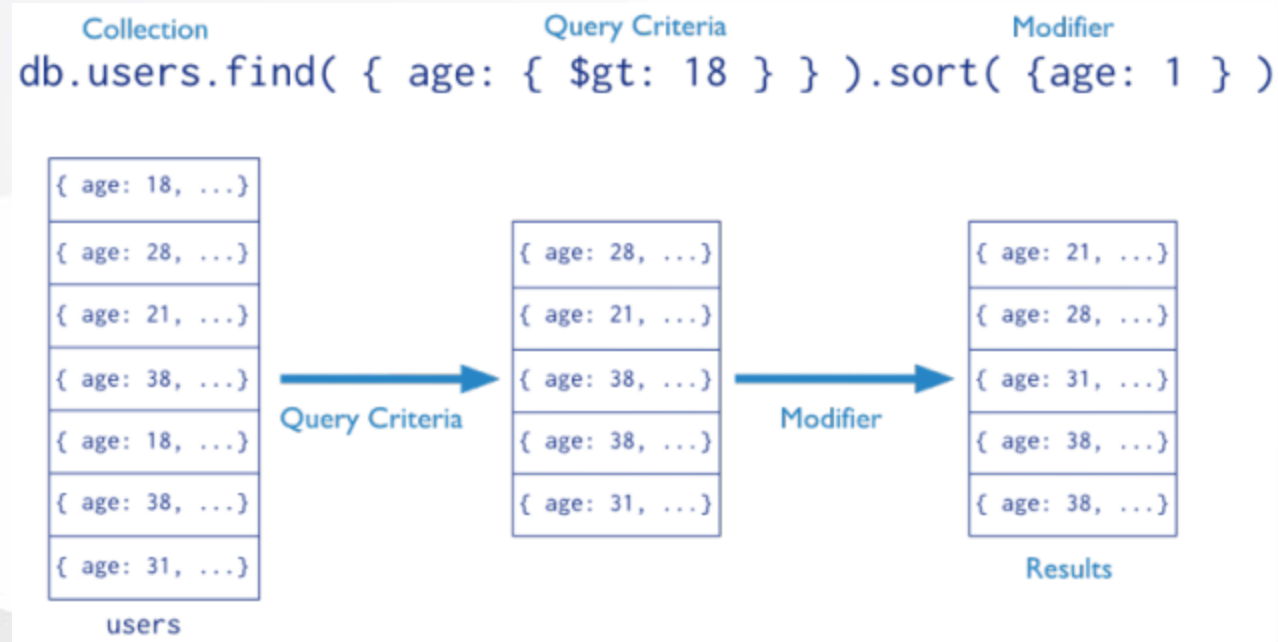




# Queries

En MongoDB las consultas se hacen sobre una colección de documentos

- Se especifican los criterios de los documentos a recuperar



# Conceptos básicos

---

Los documentos en MongoDB tienen un **esquema flexible**

- Las colecciones **no obligan a que sus documentos tengan un formato único**

Una colección puede tener varios documentos con una estructura diferente

- En la práctica los documentos de una colección comparten una estructura similar
- Todos los documentos tendrán un campo `_id`

# Relaciones entre documentos

---

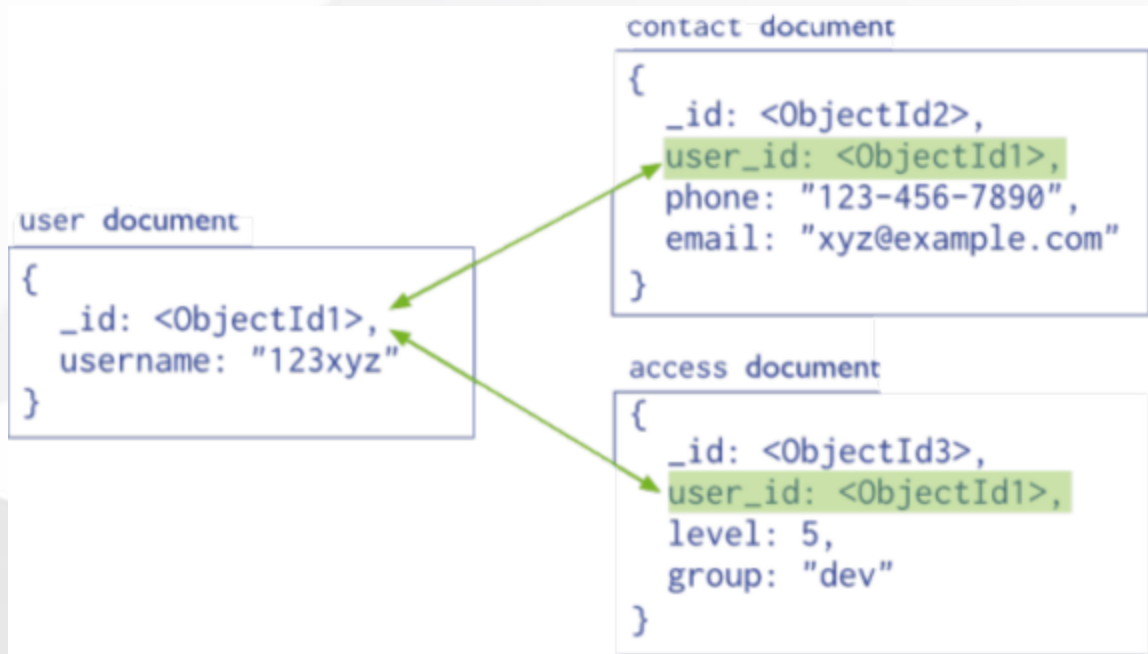
¿Cómo se representan las **relaciones** entre los datos? Dos formas:

- **Referencias** a otros documentos
- **Subdocumentos**

| Se permite (y aconseja) duplicar información

# Modelo normalizado

Ejemplo de modelo normalizado para MongoDB



# Modelo con subdocumentos

---

Ejemplo de modelo embebido para MongoDB

```
{
  _id: <ObjectId>,
  username: "123xyz",
  contact: {
    phone: "123-456-7890",
    email: "xyz@example.com"
  },
  access: {
    level: 5,
    group: "dev"
  }
}
```

Embedded sub-document

Embedded sub-document

# ¿Solución óptima?

---

La clave cuando modelamos es **balancear**:

- Las necesidades de la aplicación
- El rendimiento
- Las consultas que realizamos a los datos
- El modelo de datos está altamente relacionado con el **uso** que hacemos de los datos

# Ejemplo con Movielens

---

- Sistema de votación de películas
- Disponemos de:
  - Usuarios
  - Películas
  - Cada usuario puede votar tantas películas como desee

# Ejemplo con Movielens

- Modelo Normalizado:

Movies	Users	Ratings
<pre>{   _id: 111111,   title: "Star Wars",   director: "George Lucas",   year: 1977 }</pre>	<pre>{   _id: 999999,   nick: "juan007",   email: "juan007@gmail.com" }</pre>	<pre>{   user: 999999,   movie: 111111,   rating: 10 }</pre>
<pre>{   _id: 222222,   title: "La Vida de Brian",   director: "Terry Jones",   year: 1979 }</pre>	<pre>{   _id: 888888,   nick: "aruiz",   email: "aruiz@mtr.com" }</pre>	<pre>{   user: 999999,   movie: 222222,   rating: 7 }</pre>
		<pre>{   user: 888888,   movie: 111111,   rating: 8 }</pre>



# Ejemplo con Movielens

---

- Ventajas del modelo Normalizado:
  - Normalizado
  - Sin información duplicada
  - Un cambio en una votación se actualiza al instante
- Desventajas del modelo Normalizado:
  - Lento
  - No sigue la filosofía de MongoDB
  - Recuperar todos los votos de una película implica varias consultas

# Ejemplo con Movielens

---

- Modelo orientado a películas:

Movies		Users
{ _id: 111111, title: "Star Wars", director: "George Lucas", year: 1977, ratings: [ { _id: 999999, rating: 10 }, { _id: 888888, rating: 8 } ] }	{ _id: 222222, title: "La Vida de Brian", director: "Terry Jones", year: 1979, ratings: [ { _id: 999999, rating: 7 } ] }	{ _id: 999999, nick: "juan007", email: "juan007@gmail.com" }
		{ _id: 888888, nick: "aruiz", email: "aruiz@mtr.com" }

# Ejemplo con Movielens

---

- Ventajas del modelo orientado a películas:
  - Acceso inmediato a los votos de cada película
- Desventajas del modelo orientado a películas:
  - Recupera los votos de un usuario es más lento
  - Actualizar un voto es lento
  - Si una película tiene muchos votos el tamaño del objeto en disco puede ser demasiado grande

# Ejemplo con Movielens

- Modelo orientado a usuarios:

Movies	Users
<pre>{   _id: 111111,   title: "Star Wars",   director: "George Lucas",   year: 1977 }  {   _id: 222222,   title: "La Vida de Brian",   director: "Terry Jones",   year: 1979 }</pre>	<pre>{   _id: 999999,   nick: "juan007",   email: "juan007@gmail.com",   ratings: [     {       _id: 111111,       title: "Star Wars",       director: "George Lucas",       year: 1977,       rating: 10     },     {       _id: 222222,       title: "La Vida de Brian",       director: "Terry Jones",       year: 1979,       rating: 7     }   ] }  {   _id: 888888,   nick: "aruiz",   email: "aruiz@mtr.com",   ratings: [     {       _id: 111111,       title: "Star Wars",       director: "George Lucas",       year: 1977,       rating: 8     }   ] }</pre>

# Ejemplo con Movielens


---

- Ventajas del modelo orientado a usuarios:
  - Acceso inmediato a los votos del usuario
  - Acceso inmediato a las fichas de las películas votadas por el usuario
- Desventajas del modelo orientado a usuarios:
  - Duplica información
  - El objeto usuario puede ser muy grande si vota muchas películas
  - Un cambio en una ficha de una película implica actualizar información en los usuarios

# Ejemplo con Movielens

---


- Modelo mixto:

 center

# Ejemplo con Movielens

---

- Modelo mixto:

 center

# Ejemplo con Movielens

---

- Ventajas del modelo mixto:
  - Acceso inmediato a la información de los votos de las películas
  - Acceso inmediato a la información de los votos de los usuarios
- Desventajas del modelo mixto:
  - Mucha información duplicada
  - Objetos muy grandes



# Ejemplo con Movielens

---

- Debemos responder a las siguientes preguntas:
  - ¿Es frecuente actualizar los votos?
  - ¿Es necesario conocer quién votó cada película?
  - ¿Cada cuanto cambiamos la ficha de una película?
  - ¿Puede un usuario modificar su nick?
  - ...

# Aspectos clave

---

- MongoDB es flexible
- No existen normas para modelar la base de datos
- Solamente existen una serie de buenos consejos
- Debemos pensar en el uso de los datos
- Se puede (y se aconseja) duplicar información

# Operaciones

# Operaciones

---

MongoDB ofrece soporte para:

- Escritura (**C**reate)
- Lectura (**R**ead)
- Modificación (**U**ppdate)
- Borrado (**D**elete)

# Consultas lectura básicas

---

`db.collection.find()`: Recupera documentos de una colección

- Todas las películas:

```
db.movies.find({})
```

- Todas las estrenadas en 1995:

```
db.movies.find({year: 1995})
```

- Todas las estrenadas en 1995 y empiezan por 'A' (`i` → case insensitive):

```
db.movies.find({year: 1995, title: {$regex: "^A", options: "i"}}) # 0 $regex: /^A/i
```

- Películas estrenadas entre 1995 y 1997:

```
db.movies.find({year: {$gte: 1995}, year: {$lte: 1997}})
```

# Consultas lectura básicas

---

## Método db.collection.find() - Operadores lógicos \$and

### Sintaxis

- `{ $and: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }`

### Las películas de comedia lanzadas en 2000

- `db.movies.find({ $and: [ { genres: "Comedy" }, { year: 2000 } ] })`

## Método db.collection.find() - Operadores lógicos \$or

### Sintaxis

- `{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }`

### Las películas que sean de comedia o que hayan sido lanzadas en 2000

- `db.movies.find({ $or: [ { genres: "Comedy" }, { year: 2000 } ] })`

# Consultas lectura básicas

---

## Método db.collection.find() - Operadores lógicos \$nor

### Sintaxis

- `{ $nor: [ { <expression1> }, { <expression2> }, ... { <expressionN> } ] }`

Todas las películas que no sean de comedia y que no hayan sido lanzadas en 2000

- `db.movies.find({ $nor: [ { genres: "Comedy" }, { year: 2000 } ] })`

## Método db.collection.find() - Operadores lógicos \$not

### Sintaxis

- `{ field: { $not: { <operator-expression> } } }`

Las películas que no sean de comedia

- `db.movies.find({ genres: { $not: { $eq: "Comedy" } } })`

# Consultas lectura básicas

---

## Método db.collection.find() - Operadores de comparación \$eq

### Sintaxis

- `{ <field>: { $eq: <value> } }`

### Las películas que fueron lanzadas en el año 2016

- `db.movies.find({ year: { $eq: 2016 } })`

## Método db.collection.find() - Operadores de comparación \$gt y \$lt

### Sintaxis

- `{ field: { $gt: value } } || { field: { $lt: value } }`

### Las películas con un rating mayor a 8.0 y menor a 8.5

- `db.movies.find({ rating: { $gt: 8.0, $lt: 8.5 } })`

Se puede usar \$gte y \$lte para menor o igual y mayor o igual



# Consultas lectura básicas

---

## Método db.collection.find() - Operadores de conjuntos \$in

### Sintaxis

```
{ field: { $in: [<value1>, <value2>, ... <valueN> ] } }
```

Las películas que sean de los géneros "Comedy" o "Drama"

```
db.movies.find({ genres: { $in: ["Comedy", "Drama"] } })
```

## Método db.collection.find() - Operadores de conjuntos \$nin

### Sintaxis

```
{ field: { $nin: [ <value1>, <value2> ... <valueN> ]} }
```

Las películas que no sean de los géneros "Comedy" ni "Drama"

```
db.movies.find({ genres: { $nin: ["Comedy", "Drama"] } })
```

# Consultas lectura básicas

---

## Método db.collection.find() - Operadores de conjuntos \$all

### Sintaxis

```
{ field : { $all: [ <value1> , <value2> ... ] } }
```

Las películas que sean de los géneros "Comedy" y "Drama"

```
db.movies.find({ genres: { $all: ["Action", "Drama"] } })
```

## Método db.collection.find() - Operadores de conjuntos \$size

### Sintaxis

```
{ field: { $size: value } }
```

Todas las películas que tengan exactamente tres actores

```
db.movies.find({ actors: { $size: 3 } })
```

# Consultas lectura básicas

---

## Método `db.collection.find()` - como hace un "SELECT"

### Devolver título e `_id` de las películas de 1995

- `db.movies.find({year: 1995}, {title:1, _id: 0})`

### Devolver todos los datos menos "ratings" de las películas de 1995

- `db.movies.find({year: 1995}, {ratings:0})`

## Método `db.collection.find()` - como hace un "ORDER BY"

### Todas las películas ordenadas por año ascendente

- `db.movies.find({}).sort({year: 1})`

### Todas las películas ordenadas por año descendente

- `db.movies.find({}).sort({year: -1})`