

# UD6 – Persistència en Bases de dades documentals. MongoDB



# ÍNDEX

1	Bases de dades NoSQL.....	5
2	Introducció a MongoDB.....	6
2.1	On es pot utilitzar MongoDB?.....	7
2.2	Com funciona MongoDB?.....	7
2.3	Qui usa MongoDB?.....	8
2.4	Avantatges de MongoDB.....	9
2.5	Inconvenients de MongoDB.....	11
3	Estructura d'una BD MongoDB.....	12
3.1	Col·leccions i Documents.....	12
3.1.1	Tipus de dades.....	14
3.2	Camp _id.....	16
3.3	Esquema de dades - Relacions en el model.....	18
3.3.1	One-to-one (UN a UN).....	18
3.3.2	One-to-many incrustat (UN a MOLTS).....	18
3.3.3	One-to-many amb referències (UN a MOLTS).....	20
4	Instal·lació del Sistema Gestor de Bases de Dades MongoDB.....	21
5	Connexió amb MongoDB Compass al servidor de bases de dades.....	24
5.1	Creació d'una base de dades i una col·lecció.....	25
5.2	Importació de dades d'un document JSON.....	27
5.3	mongoimport.....	29
5.3.1	Paràmetres avançats per a mongoimport.....	30
6	Comandos bàsics.....	32

## UD6 - Persistència en BBDD documentals. MongoDB

6.1	Insert()	32
6.2	find()	32
6.3	Exemples	34
7	Pràctica 1	37
8	Connexió a MongoDB desde Java	40
8.1	Creació i configuració projecte Java amb connexió amb MongoDB	40
8.2	Creem la classe “ObtindrePeliculesMongoDB”	42
8.2.1	Llibreries de classes per a importar	44
8.3	Establint una connexió	45
8.3.1	Connectar-se a una única instància de MongoDB	45
8.4	Obtenint una Base de dades	46
8.5	Obtenint una Col·lecció	47
9	Consultes a la base de dades MongoDB	48
9.1	Classe Document	48
9.2	Obtindre el primer document d'una col·lecció	49
9.3	Obtindre tots els documents d'una col·lecció	50
9.4	Classe Filters: especificar un filtre de consulta	51
9.4.1	Obtindre un document únic que coincidisca amb un filtre	53
9.4.2	Obtindre tots els documents que coincideixen amb un filtre	54
9.5	Contar documents en una col·lecció	55
9.6	Buscar patrons en els camps dels documents	55
9.7	Consultes sobre arrays	57
9.7.1	Buscar elements dins d'un array	57

9.7.2	Contar els elements que conté un array.....	57
9.8	Projeccions.....	58
9.9	Ordenant els resultats.....	59
10	Inserció de documents a la base de dades MongoDB.....	60
10.1	Crear un document.....	60
10.2	Inserir un Document.....	61
10.3	Inserir Múltiples Documents.....	61
11	Actualització de documents a la base de dades MongoDB.....	62
11.1	Actualitzar un únic document.....	62
11.2	Actualitzar múltiples documents.....	63
12	Eliminació de documents a la base de dades MongoDB.....	63
12.1	Eliminar un únic document que compleix una condició.....	63
12.2	Eliminar tots els documents que compleixen una condició.....	64

## 1 Bases de dades NoSQL

A pesar que les bases de dades relacionals han satisfet les necessitats de la immensa majoria d'usuaris en les últimes dècades, no podríem dir que són infal·libles en totes les solucions.

Si bé la quota de mercat de les solucions NoSQL no està a l'altura de les relacionals, les majors empreses d'Internet ja tenen els seus propis sistemes basats en esta tecnologia: Amazon amb la seua **DynamoDB** o LinkedIn amb **RedIs**.

El contingut d'este tema que comencem girarà entorn a una d'estes bases de dades NoSQL, concretament de la base de dades documental MongoDB.



## 2 Introducció a MongoDB

El nom de “MongoDB” ve de la paraula anglesa “humongous” (enorme, extraordinàriament llarg).

**MongoDB** és una **base de dades documental** que ofereix una gran escalabilitat i flexibilitat, i un model de consultes i indexació avançat.

Presenta les següents **característiques**:

- MongoDB **emmagatzema dades en documents JSON flexibles**, és a dir, cada document pot contindre diferents camps i les estructures de dades es poden anar modificant.
- El model de documents **concorda amb els objectes del codi de l'aplicació**, la qual cosa facilita treballar amb dades.
- Les **consultes ad hoc, la indexació i l'agregació en temps real** permeten accedir a les dades i analitzar-los amb gran eficàcia.
- És una **base de dades distribuïda**, per la qual cosa és fàcil d'usar i proporciona una elevada disponibilitat, escalabilitat horitzontal i distribució geogràfica.
- MongoDB és una **base de dades de codi obert i d'ús gratuït**.

```
{
  "_id" : ObjectId("5b1245d632b2a4cd81"),
  "referencia" : "123456",
  "nom" : "Motxilla portatil 15 polsades"
  "marca" : "",
  "imatgens" : {
    "gran" : "/imatgens/productes/motxilles/123456_gran.png",
    "miniatura" : "/imatgens/productes/motxilles/123456_xicoteta.png"
  },
  "color" : "negre",
  "stock" : 12,
  "categoria" : [
    "portatil",
    "motxilles",
    "viatges"
  ]
  "preu" : NumberDecimal("35,52"),
  "opinions" : [] ,
  "likes" : 122,
  "tweets" : 16,
  "tipus" : "motxilla"
}
```

*Exemple “motxilla.json”*

## 2.1 On es pot utilitzar MongoDB?

Encara que se sol dir que les bases de dades NoSQL tenen un àmbit d'aplicació reduït, MongoDB es pot utilitzar en molts dels projectes que desenvolupem en l'actualitat.

Qualsevol aplicació que necessite emmagatzemar dades semiestructurades pot usar MongoDB. És el cas de les típiques aplicacions CRUD o de molts dels desenvolupaments web actuals.

Això sí, encara que les col·leccions de MongoDB no necessiten definir un **esquema**, és important que dissenyem la nostra aplicació per a seguir-ne un. Haurem de pensar si necessitem normalitzar les dades, desnormalitzar-les o utilitzar una aproximació híbrida. Estes decisions poden afectar el rendiment de la nostra aplicació. En definitiva l'esquema el definixen les consultes que anem a realitzar amb més freqüència.

MongoDB és especialment útil en entorns que requereixen escalabilitat. Amb les seues opcions de **replicació** i **sharding**, que són molt senzilles de configurar, podem aconseguir un sistema que escale horitzontalment sense massa problemes.

El **Sharding** és un mètode per a distribuir dades a través de múltiples màquines. MongoDB utilitza el sharding per a suportar desplegaments amb conjunts de dades molt grans i operacions d'alt rendiment.

## 2.2 Com funciona MongoDB?

MongoDB ve de sèrie amb una consola des de la qual podem executar els diferents comandos. Esta consola està construïda sobre JavaScript, per la qual cosa les consultes es realitzen utilitzant aqueix llenguatge. A més de les funcions de MongoDB, podem utilitzar moltes de les funcions pròpies de JavaScript. En la consola també podem definir variables, funcions o utilitzar bucles.

Si volem usar el nostre llenguatge de programació favorit, existeixen drivers per a un gran nombre d'ells. Hi ha **drivers oficials** per a C#, **Java**, Node.js, PHP, Python, Ruby, C, C++, Perl o Scala. Encara que estos drivers estan suportats per MongoDB , no

tots estan en el mateix estat de maduresa. Si volem utilitzar un llenguatge concret, és millor revisar els drivers disponibles per a comprovar si són adequats per a un entorn de producció.

MongoDB està escrit en C++, encara que les consultes es fan passant objectes JSON com a paràmetre. És una cosa bastant lògica, donat que els propis documents s'emmagatzemen en BSON.

## 2.3 Qui usa MongoDB?

- Internet de les coses (IoT), grup industrial Bosch.
- Visualització geoespacial d'elements d'una ciutat en temps real (Boston city).
- Gestió de continguts, cas de Sourceforge .
- Aplicacions mòbils, com a compra de viatges per Expedia.
- Videojocs com a FIFA online 3.
- Logs d'esdeveniments, cas de Facebook per a recollir anuncis accedit.
- Alguns usuaris coneguts de MongoDB són: Ebay, Expedia, Orange, Barclays, Adobe, Telefónica...

Veure més en:

<https://www.mongodb.com/use-cases>

<https://www.mongodb.com/who-uses-mongodb>

**EXEMPLE:** “250.000 rutas GPS de BiciMad”

<https://www.microsiervos.com/archivo/mundoreal/250000-rutas-gps-bicimad-bicicletas-alquiler-madrid.html>



## 2.4 Avantatges de MongoDB

La filosofia de MongoDB és mantindre el major número de funcionalitats possibles, permetent al mateix temps un escalat horitzontal i facilitant la vida dels desenvolupadors. Se situa entre la potent però ineficient **base de dades relacional** i el sistema d'emmagatzematge de **valors-clau** d'alt rendiment però simple.

### Rendiment

MongoDB és una base de dades dissenyada per a l'emmagatzematge i consulta de grans quantitats de dades (Big Data), dirigida a aplicacions de xarxes socials com Facebook.

Obté el seu rendiment principalment gràcies a un disseny basat en valors clau i fàcil d'ampliar. Sent una base de dades NoSQL, MongoDB utilitza el document com a unitat bàsica d'emmagatzematge.

Un **document** és un simple objecte similar a JSON, anomenat BSON en MongoDB, que és només un BLOB (Binary Large Object).

Per exemple, una entrada de blog consta de títol, contingut i comentaris:

- En el model relacional, el comentari s'emmagatzemarà com una taula individual i es recuperarà unint la taula de missatges i la taula de comentaris.
- En el model de document, es graven com un sol document. Es tracten com un sol objecte. En realitzar la consulta, s'obté tot eixe document, sense referència a altres documents, que un document pot ser identificat per un identificador.

Per tant, obtenir un document és una consulta de valor clau, no una consulta relacional. La consulta de valors clau és molt més ràpida que la consulta relacional.

### Escalabilitat

El segon avantatge de rendiment que ofereix MongoDB és **l'escalabilitat**. En MongoDB, és molt fàcil escalar horitzontalment. És a dir, afegir més màquines a mesura que les dades i el trànsit creixen i mantindre la velocitat de resposta i la disponibilitat.

Per a fer més eficients les consultes, la base de dades MongoDB suporta indexació. S'implementa amb Arbres Binaris, i l'índex pot ser definit en camp únic o múltiples camps, igual que els índex en les base de dades MySQL.

### El model documental

El model documental també facilita el desenvolupament d'aplicacions. El Document no té els conceptes de taules, files, SQL... L'esquema de la base de dades es crea sobre la marxa. Bàsicament, es pot afegir qualsevol tipus de camp a una col·lecció de MongoDB en qualsevol moment.

Si tenim moltes dades no estructurades, el model de document és una molt bona opció. Per exemple, les estadístiques de trànsit del lloc web, estadístiques de publicitat, etc. Dades com estes tenen una grandària enorme, estan feblement relacionades entre si i no necessiten una consistència del 100%.

MongdoDB és bo per a emmagatzemar i processar dades com les de l'exemple anterior. De fet, MongoDB va ser desenvolupat en DoubleClick, que realitzava un seguiment de les activitats publicitàries online, ara adquirida per Google per a servir a la seua xarxa d'editors d'Adsense.

### Esquema flexible

En realitat, no hi ha cap esquema en MongoDB, el document pot tindre qualsevol nombre de camps, els camps poden ser afegits al document existent en qualsevol moment, dinàmicament. A més, un mateix camp pot ser de distints tipus.

MongoDB és capaç de representar models rics de dades amb el format de dades BSON, igual que una base de dades d'esquema estricte com MySQL.

Este avantatge fa possible el desenvolupament àgil d'aplicacions, perquè a diferència del model relacional, les dades de JSON poden encaixar quasi perfectament en el model Orientat a Objectes que la majoria dels llenguatges de programació suporten hui dia.

En el model de base de dades tradicional es necessita una ferramenta de Mapeig Objecte-Relacional com Hibernate per a tractar el desajustament entre la Programació Orientada a Objectes i el model relacional.



## 2.5 Inconvenients de MongoDB

Tot i que hem vist que MongoDB presenta molts avantatges respecte al model relacional, també en trobem alguns inconvenients.

### No suporta transaccions

Per a aconseguir l'alt rendiment i l'escalabilitat, **MongoDB no dóna suport a les transaccions**. Això fa que siga molt fàcil d'escalar horitzontalment, ja que utilitza hardware molt econòmic per a equilibrar la càrrega.

MongoDB és una **bona elecció** si tenim **moltes dades**, però la **relació** entre elles és **feble**, per exemple, un flux d'esdeveniments independents.

**No és apte** per a **dades fortament relacionades** com per exemple la informació de comptes bancaris. Les transferències impliquen modificar informació en varies entitats, i o bé es fan totes les modificacions, o bé no es fa res. Els RDBMS ho fan amb transaccions.

Per a intentar superar esta mancança, MongoDB suporta operacions atòmiques com increment i decrement atòmic, **findAndModify**. No obstant això, només la modificació d'un sol document és atòmica, però les operacions que impliquen múltiples documents no són atòmiques.

### No suporta la CONCATENACIÓ (JOIN)

Com que el document conté, a priori, tot el que necessites, **no hi ha suport a les operacions de Concatenació (Join)** a MongoDB. Esta és també la compensació per a poder escalar fàcilment horitzontalment. L'única manera de realitzar la unió seria fent múltiples consultes.

### Limitació de memòria RAM

MongoDB utilitza un arxiu de memòria mapejada, i deixa que el sistema operatiu s'encarregue de la caché. La grandària de la seua base de dades està limitada per la memòria virtual proporcionada pel sistema operatiu i el maquinari, encara que no sòl ser un problema degut a les capacitats de memòria dels sistemes actuals.



## 3 Estructura d'una BD MongoDB

Si tenim moltes dades no estructurades i de les quals es realitzaran moltes consultes, però poques modificacions, el model del document és una bona opció.

Per exemple, les estadístiques de trànsit d'un lloc web, xarxes socials, publicitat... Emmagatzemar gran quantitat d'informació, feblement relacionada i que no és necessària la consistència del 100%.

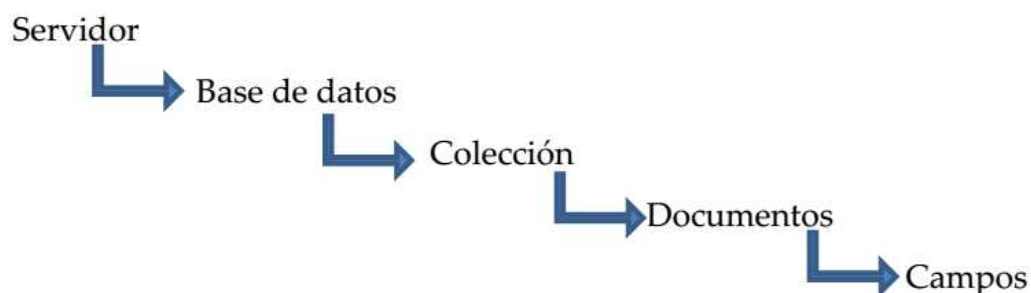
MongoDB és una **Base de dades NoSQL orientada a documents** en format  **BSON**  Binary Javascript Object Notation), similar a JSON. Este és el format que s'utilitza intercanvi de dades per a emmagatzematge i transferència de documents en MongoDB. Es tracta d'una representació binària d'estructures de dades i mapes, dissenyada per a ser més lleuger i eficient que JSON, (Javascript Object Notation), amb algunes **particularitats**:

- Tipus de dades addicionals.
- Dissenyat per a ser més eficient en espai, encara que a vegades un JSON pot ocupar menys que un BSON.
- Dissenyat perquè les cerques siguin més eficients.

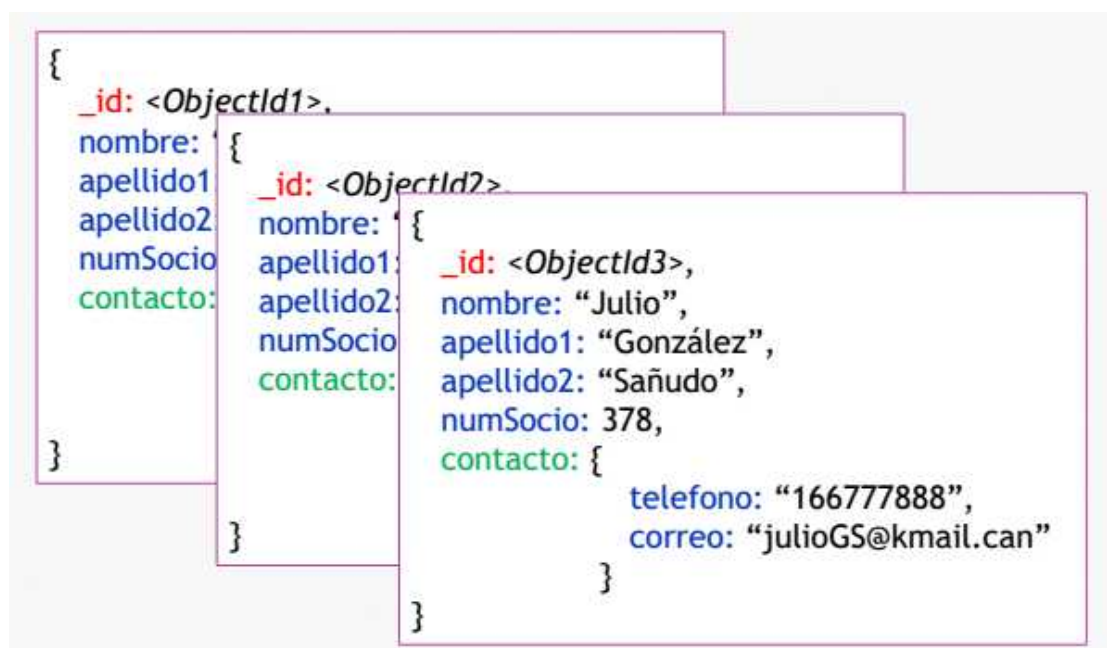
### 3.1 Col·leccions i Documents

Les bases de dades estan formades per **Col·leccions**, i al mateix temps, cada servidor pot tindre tantes bases de dades com l'equip ho permeti.

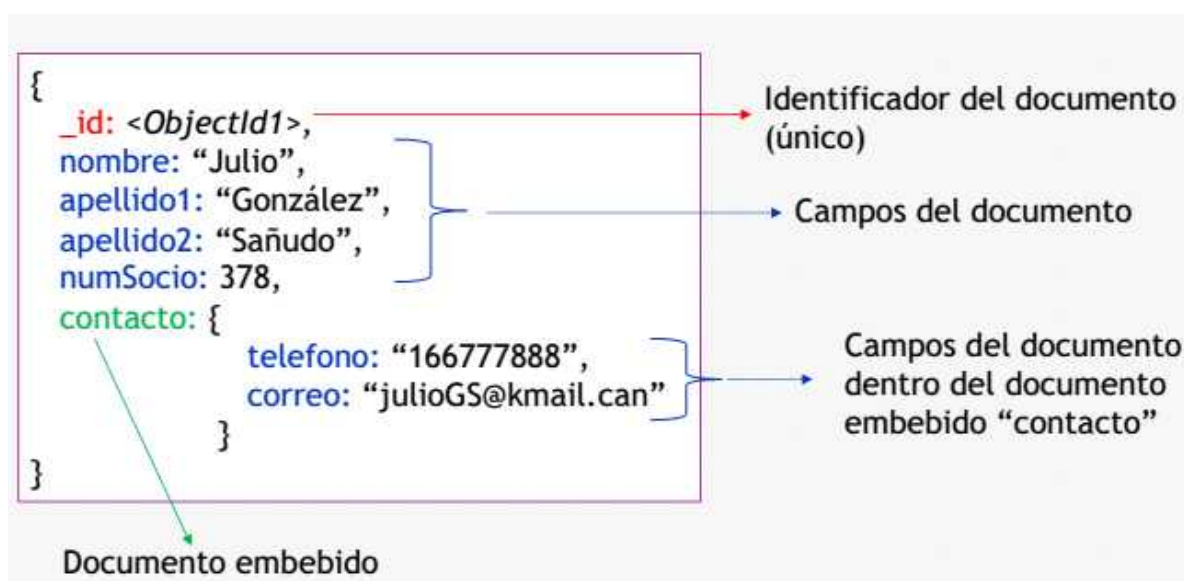
En MongoDB, cada registre o conjunt de dades es denomina **document**, que poden ser agrupats en **col·leccions**, (equivalent a les taules de les bases de dades relacionals però sense estar sotmesos a un esquema fix).



### Col·lecció:



### Document:



Un **registre** de MongoDB és un document, una estructura de dades composta per **parells de camps i de valor**. Estos documents són similars als objectes **JSON**. Els valors dels camps poden incloure:

- Altres documents.
- Matrius (arrays).
- Conjunts de documents.

En MongoDB, documents emmagatzemats en una col·lecció han de tindre un **camp \_id únic**, que actua com una clau principal.

### 3.1.1 Tipus de dades

**MongoDB**, a través de JSON, pot utilitzar els següents tipus:

- **String**: guardats en UTF-8. Van sempre entre dobles cometes.
- **Number**: números. En guardar-se en BSON poden ser de tipus byte, int32, int 64 o double.
- **Boolean**: amb valor true o false.
- **De data**:
  - **Date**: enter de 64 bits que representa el nombre de mil·lisegons des de l'1 de gener de 1970.
  - **Timestamp**: enter de 64 bits, en el qual els primers 32 bits representen els segons passats des de l'1 de gener de 1970, i els altres 32 bits són ordinals incrementals. En una instància mongod, cada valor de timestamp és únic.
- **Especials**:
  - **Array**: emmagatzema un conjunt d'elements de qualsevol tipus. van entre claudàtors [] i poden contindre d'1 a N elements, que poden ser de qualsevol dels altres tipus.
  - **ObjectId**: tipus de dada única, principalment utilitzat per a donar valor al camp \_id dels documents.
  - **Javascript**: codi javascript.
  - **Null**: valor nul.
  - **Documents**: un document en format JSON pot contindre altres documents embeguts que incloguen més documents o qualsevol dels tipus anteriorment descrits.

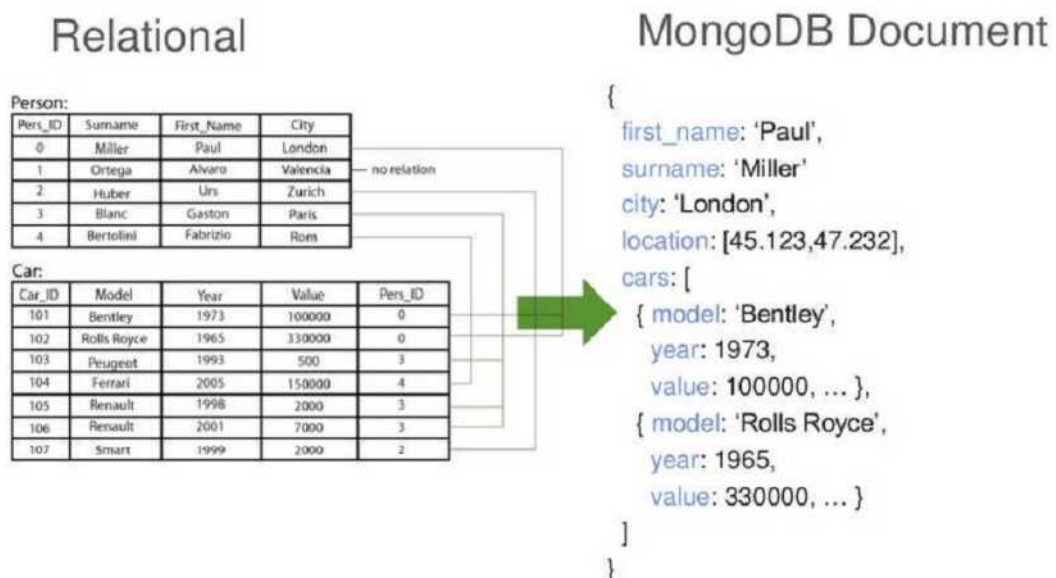


```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

← field: value  
← field: value  
← field: value  
← field: value

La jerarquia principal de MongoDB ve donada pels elements presentats a dalt. La diferència amb els SGBDR, (Sistemes Gestors de Bases de dades Relacionals), és que no utilitzem taules, files ni columnes com allí ho féiem, sinó que utilitzem documents amb diferents estructures.

Un conjunt de Camps formaran un Document, que en cas d'associar-se amb uns altres formarà una Col·lecció.



En MongoDB, com ja s'ha comentat adés, no existeix un esquema estàndard per a treballar amb les dades, però això no significa que anem a tindre una quantitat ingent de dades difícils de relacionar.

De fet, la majoria de les vegades treballarem amb documents estructurats, només que no seguiran el mateix esquema tots ells, sinó que cadascun podrà tindre un propi si resulta apropiat treballar amb ells.

Imaginem que tenim una col·lecció a la qual anomenem Persones. Un document podria a emmagatzemar-se la següent manera:

```
{
  Nom: "Javi",
  Cognoms: "Martínez Campos",
  Edat: 22,
  Aficions: ["futbol", "tennis", "ciclisme"],
  Amics: [
    { Nom: "Maria",
      Edat: 22 },
    { Nom: "Lluís",
      Edat: 28 }
  ]
}
```

- Les **claus {}** delimiten documents embeguts és a dir, un conjunt de parells clau - valor.
- Els **[]** delimiten un array, llista de valors associats a una clau.

El document anterior és un clàssic document JSON. Té strings, arrays, subdocuments i números. En la mateixa col·lecció podríem guardar un document com este:

```
{
  Nom: "Luis",
  Estudis: "Administració i Direcció d'Empreses",
  Amics: 12
}
```

Este document no segueix el mateix esquema que el primer. Té menys camps, algun camp nou que no existix en el document anterior i fins i tot un camp de diferent tipus.

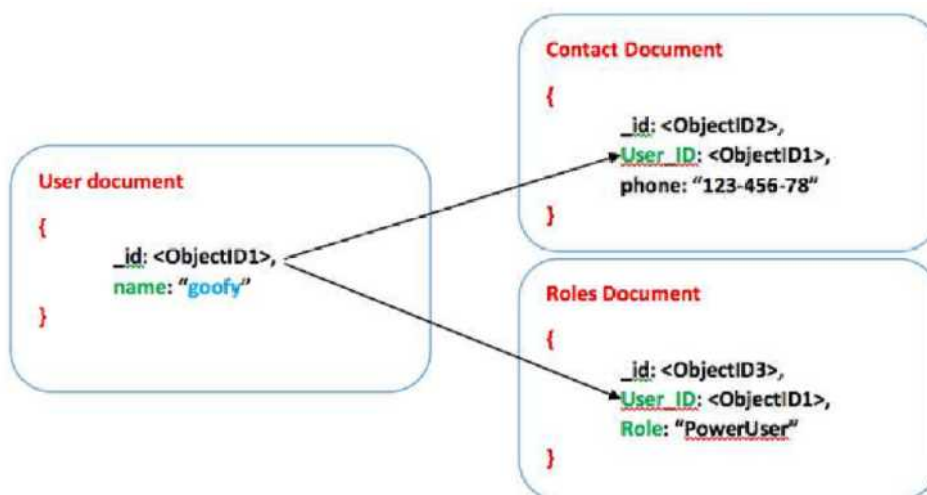
Açò, que seria impensable en una base de dades relacional, és una cosa totalment vàlida en MongoDB.

### 3.2 Camp *\_id*

El camp especial `_id`, creat amb `ObjectId`. Este objecte, que és en realitat tractat com un String, es crea automàticament en tots els documents que inserim en la nostra base de dades i té un valor únic.

- Podem especificar nosaltres el valor que vulguem, però haurem de tindre en compte que **no pot haver dos documents amb el mateix \_id**.
- El camp **\_id** es crea automàticament amb la funció **ObjectId**, però podem inserir valors únics generats amb esta funció en qualsevol camp del nostre document.
- **\_id** és un número hexadecimal de 12 bytes únic per a cada document en una col·lecció.
- L'ID dels registres és preferible introduir-lo manualment perquè si deixem que el propi MongoDB ens l'autogeneri no ho fa com un número enter normal i a l'hora de buscar-lo des d'una aplicació pot resultar complicat.

Així que es recomanable generar els ID dels registres en el servidor amb un generador de ids únic i afegir-lo als teus registres.



### 3.3 Esquema de dades - Relacions en el model

Com hem dit adés, en MongoDB l'esquema de les dades és flexible. Correspon al desenvolupador decidir com implementar-lo segons els requisits que es tinguen.

Es poden definir 3 **tipus de relacions** que vorem a continuació.

- **One-to-one**
- **One-to-many incrustat**
- **One-to-many amb referències**

#### 3.3.1 One-to-one (UN a UN)

En este tipus de relació, un dels documents sol incloure's dins d'un altre. Imaginem que emmagatzemem les dades dels socis d'un gimnàs juntament amb la seua direcció, que es compon dels seus propis camps. La solució passaria per incloure la direcció com un subdocument del document principal.

```
{ _id: <ObjectId>,
  nombre: "Julio",
  apellido1: "González",
  ...
  dirección: {
    calle: "Avenida de la República",
    numero: 678
    ...
  }
}
```

#### 3.3.2 One-to-many incrustat (UN a MOLTS)

Els documents de la relació s'inclouen dins d'un altre en una estructura de tipus array. Seguint amb l'exemple anterior, imaginem que els socis poden tindre diverses direccions:

```
{ _id: <ObjectId>,  
  nombre: "Julio",  
  apellido1: "González",  
  ...  
  direcciones: [{  
    calle: "Avenida de la República",  
    numero: 678  
    ...},  
    {calle: "Avenida de la República",  
     numero: 678  
     ... } ] }
```

A voltes, pot ser molt pesat emmagatzemar matrius (arrays) amb totes les dades dels subdocuments. Imaginem que volem emmagatzemar llibres juntament amb la referència als editors (molts).

Si pensem en un exemple en el qual podríem emmagatzemar informació sobre llibres d'un editor:

- a) Una opció seria els llibres en una matriu (array) com subdocuments dels documents dels editors:

```
{ _id: "edicionesSotileza",  
  ciudad: "Santander",  
  ...  
  libros: [{  
    nombre: "Historia de Cantabria",  
    ISBN: "678789789"  
    ...},  
    {nombre: "El pleito de los nueve valles",  
     ISBN: "2671982093"  
     ... } ] }
```

El problema de l'anterior solució és que la matriu (array) podria créixer en excés, amb el que les cerques seria molt ineficients.

- b) Podríem pensar com a solució incloure als editors en els llibres:

```
{_id: "HdC1",  
  nombre: "Historia de Cantabria",  
  ISBN: "678789789"  
  ...  
  editor: { nombre: "edicionesSotileza",  
            ciudad: "Santander",  
            ...}  
}
```

El problema de l'anterior solució és que es repetirien amb freqüència les dades dels editors, ocupant una gran quantitat d'espai.

### 3.3.3 One-to-many amb referències (UN a MOLTS)

S'utilitzen referències al `_id` dels documents relacionats, en comptes d'incloure'ls per complet.

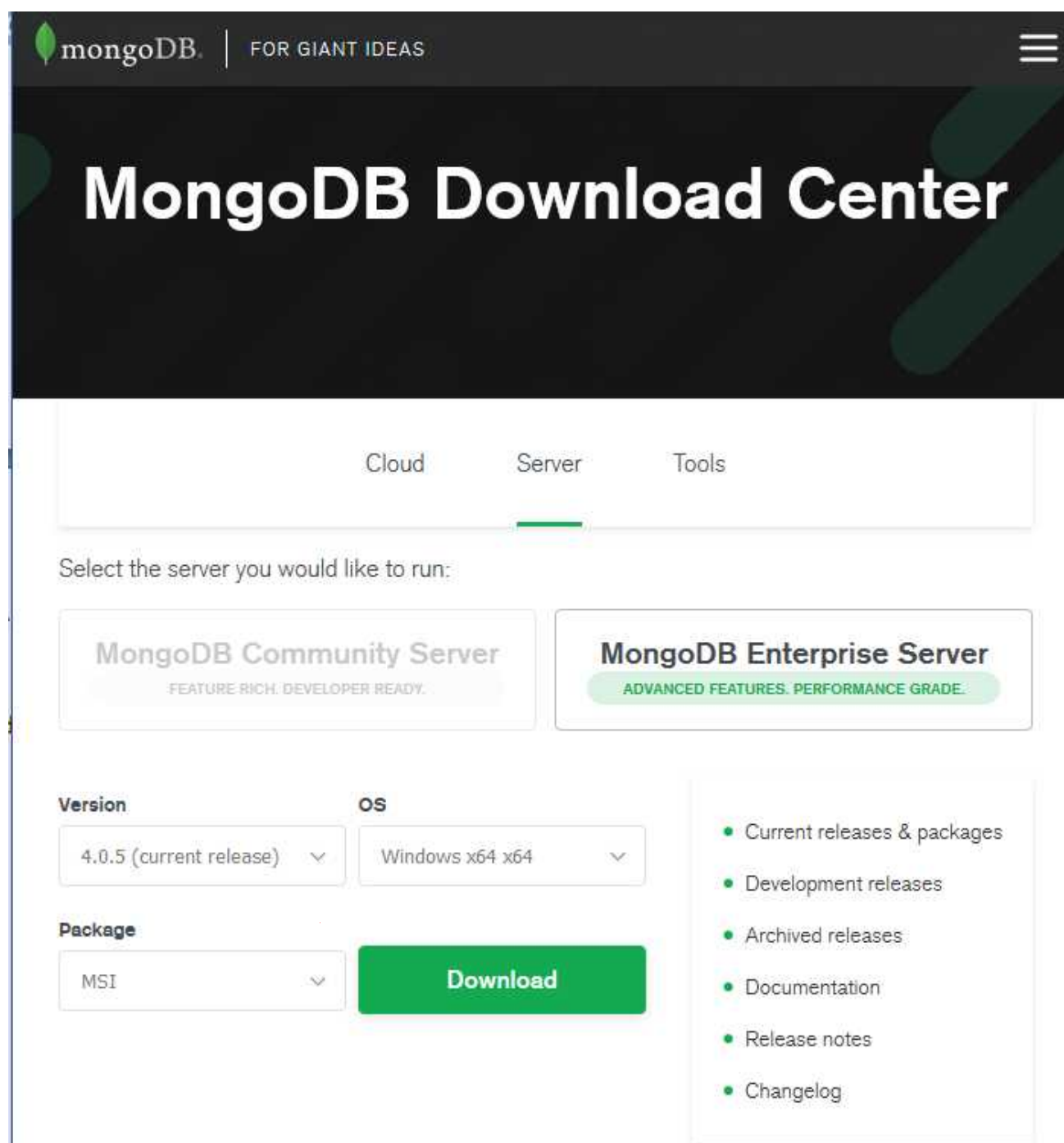
```
{_id: "edicionesSotileza",  
  ciudad: "Santander",  
  ...}
```

```
{_id: "HdC1",  
  nombre: "Historia de Cantabria",  
  ISBN: "678789789"  
  ...  
  editor: "edicionesSotileza"}
```

## 4 Instal·lació del Sistema Gestor de Bases de Dades MongoDB.

En primer lloc, caldrà anar a la pàgina oficial de MongoDB per tal de descarregar-nos el servidor MongoDB en la seua versió Enterprise ([descarregueu-vos la versió 4.0](https://www.mongodb.com/download-center/enterprise))

<https://www.mongodb.com/download-center/enterprise>



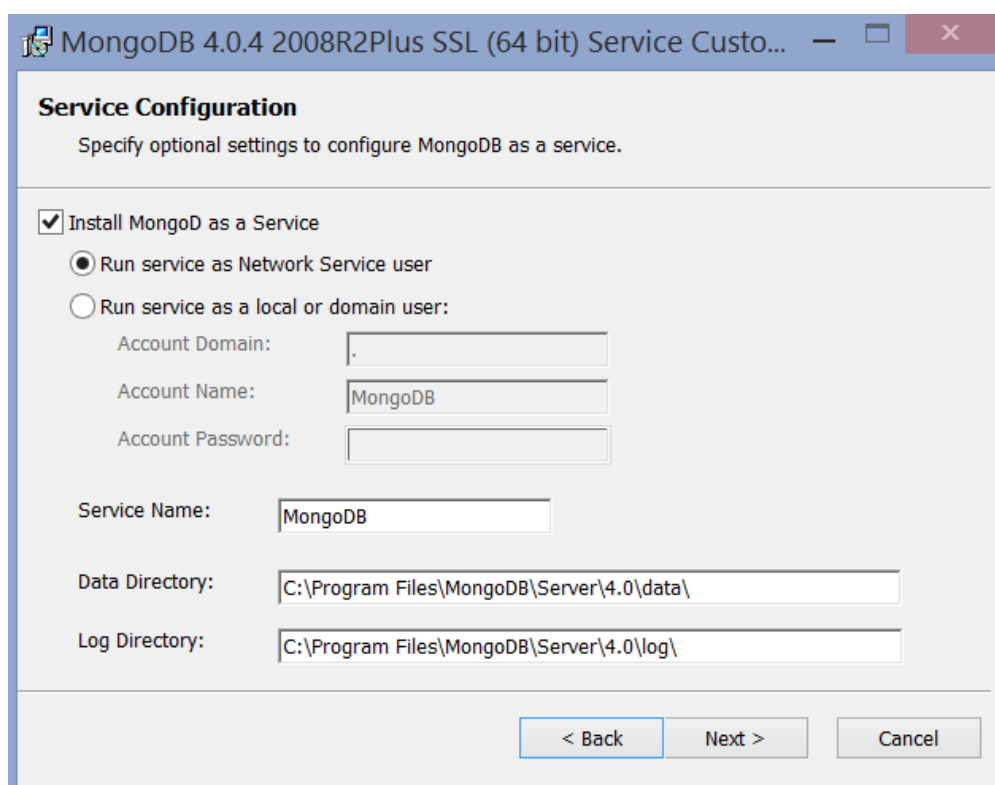
The screenshot shows the MongoDB Download Center interface. At the top, the MongoDB logo and tagline "FOR GIANT IDEAS" are visible. Below the header, the "MongoDB Download Center" title is prominently displayed. A navigation bar includes "Cloud", "Server" (which is selected), and "Tools". Under the "Server" tab, users are prompted to "Select the server you would like to run:". Two options are presented: "MongoDB Community Server" (labeled "FEATURE RICH. DEVELOPER READY.") and "MongoDB Enterprise Server" (labeled "ADVANCED FEATURES. PERFORMANCE GRADE."). Below these, there are dropdown menus for "Version" (set to "4.0.5 (current release)") and "OS" (set to "Windows x64 x64"). A "Package" dropdown is set to "MSI". A large green "Download" button is positioned next to these selections. To the right, a sidebar lists additional resources: "Current releases & packages", "Development releases", "Archived releases", "Documentation", "Release notes", and "Changelog".

## UD6 - Persistència en BBDD documentals. MongoDB

Un volta descarregat el paquet MSI (Microsoft Windows Installer), l'executem com a Administradors del sistema i continuem amb el procés d'instal·lació. Hem de triar la instal·lació completa per disposar de totes les funcionalitats que té esta versió.



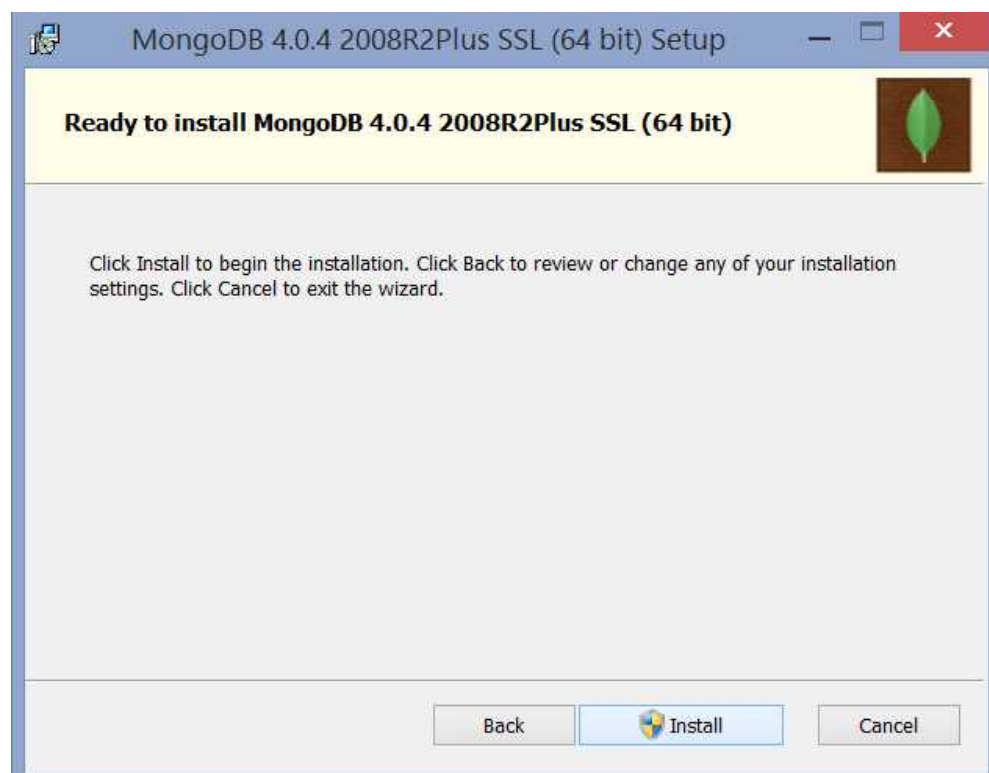
En aplegar a la pantalla on es tria la configuració de MongoDB, cal triar l'opció que l'instal·larà com a servici deixant totes les opcions per defecte.





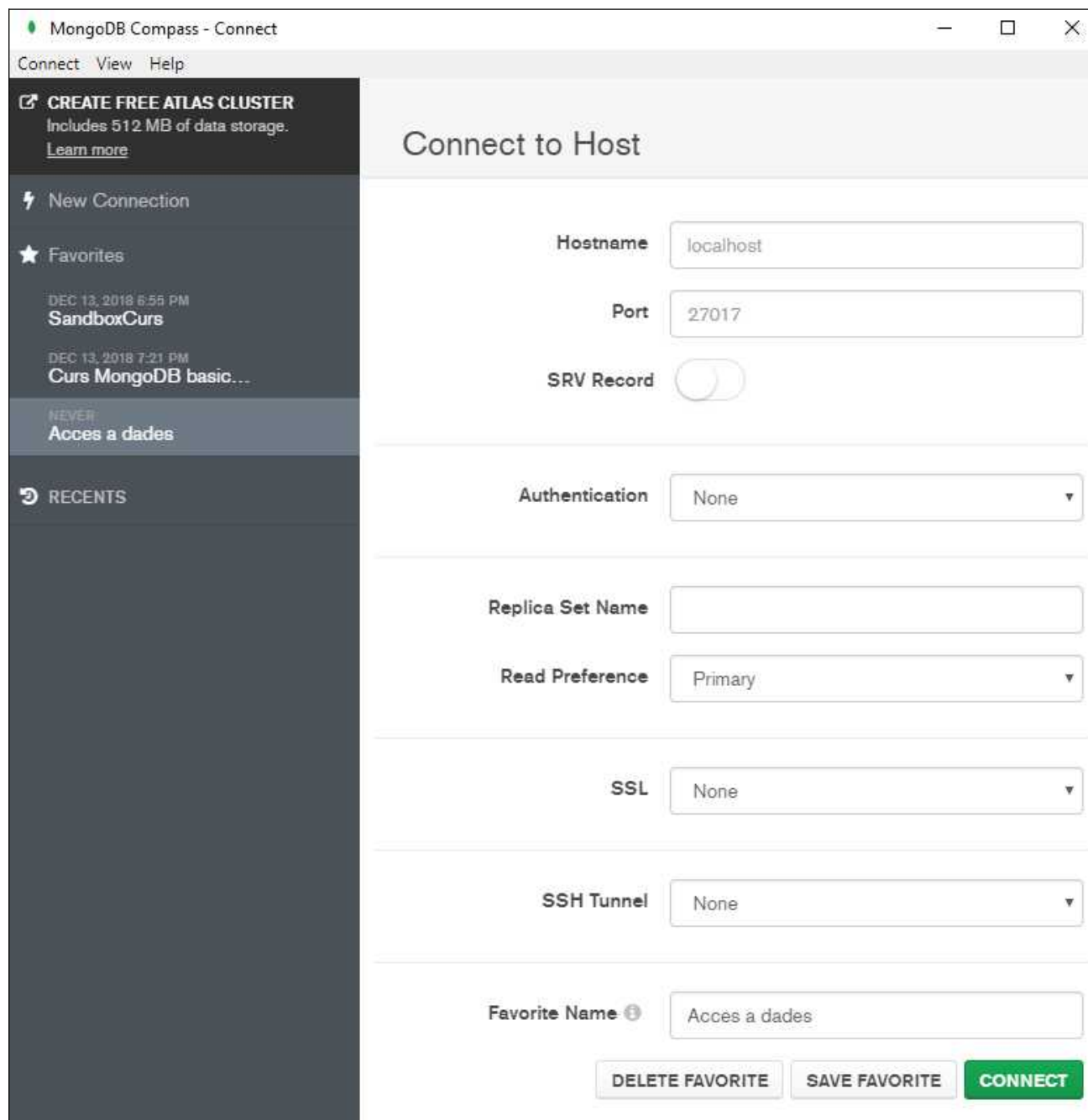
## UD6 - Persistència en BBDD documentals. MongoDB

Durant el procés d'instal·lació, l'assistent ens preguntarà si volem instal·lar l'aplicació **MongoDB Compass** (interfície gràfica d'usuari oficial de MongoDB). Marquem la casella per tal que ho faci.



## 5 Connexió amb MongoDB Compass al servidor de bases de dades

Obrim l'aplicació que acabem d'instal·lar en el punt anterior (MongoDB Compass) i obtindrem la següent pantalla:



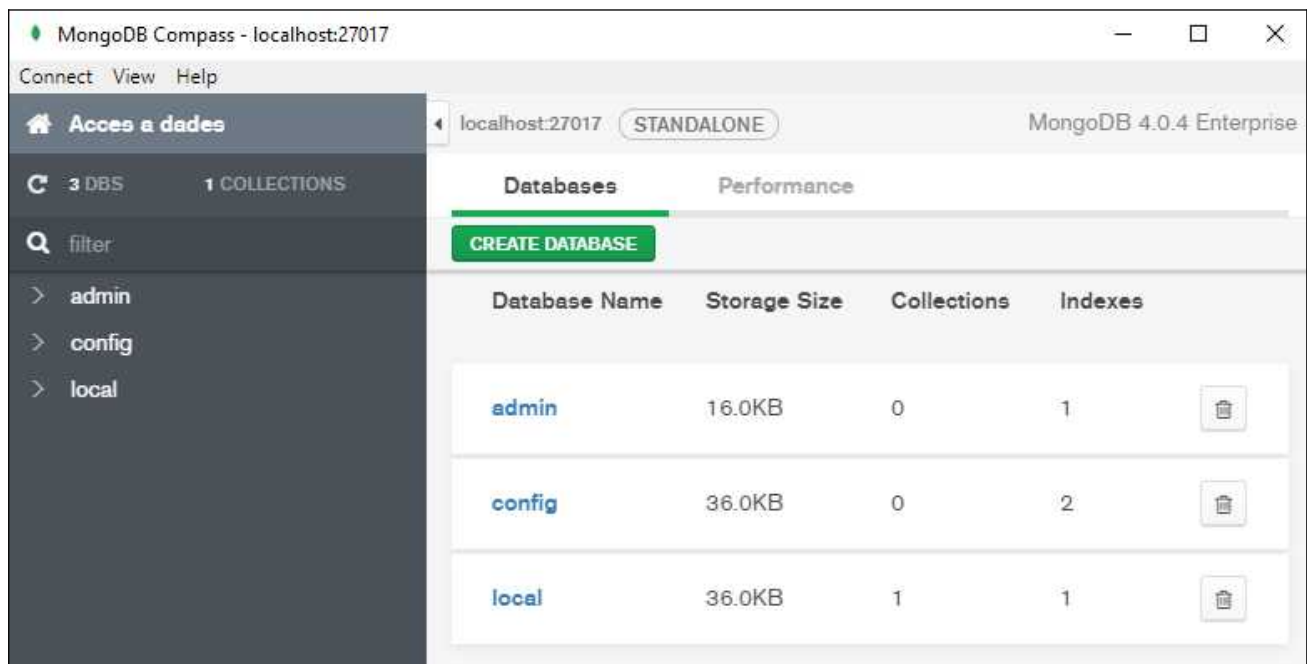
The screenshot shows the 'MongoDB Compass - Connect' window. On the left is a sidebar with options: 'CREATE FREE ATLAS CLUSTER' (with a link to 'Learn more'), 'New Connection', 'Favorites' (listing 'SandboxCurs' and 'Curs MongoDB basic...'), 'NEVER' (with a link to 'Acces a dades'), and 'RECENTS'. The main area is titled 'Connect to Host' and contains the following fields and controls:

- Hostname:** A text input field containing 'localhost'.
- Port:** A text input field containing '27017'.
- SRV Record:** A toggle switch currently turned off.
- Authentication:** A dropdown menu set to 'None'.
- Replica Set Name:** An empty text input field.
- Read Preference:** A dropdown menu set to 'Primary'.
- SSL:** A dropdown menu set to 'None'.
- SSH Tunnel:** A dropdown menu set to 'None'.
- Favorite Name:** A text input field containing 'Acces a dades'.

At the bottom right, there are three buttons: 'DELETE FAVORITE', 'SAVE FAVORITE', and a green 'CONNECT' button.

En el camp “Favorite Name” posem el nom “**Accés a dades**” i polsem el botó “SAVE FAVORITE”, de manera que cada volta que entrem amb el Compass tinguem un accés directe a la connexió al servidor MongoDB que tenim instal·lat en el nostre equip.

Una volta creat, polsem en “CONNECT” i ens durà a la interfície del nostre servidor de bases de dades:



En esta vista podem vore les Bases de Dades que es creen per defecte al servidor MongoDB:

- admin
- config
- local

### 5.1 Creació d'una base de dades i una col·lecció

En el cas de les nostres pràctiques, anem a utilitzar una base de dades de **pel·lícules**.

Per a crear la nostra pròpia base de dades, polsarem el botó “CREATE DATABASE” i posem el nom de la BD i de la primera col·lecció que volem crear.

## UD6 - Persistència en BBDD documentals. MongoDB

### Create Database

**Database Name**

**Collection Name**

☐ Capped Collection ⓘ  
☐ Use Custom Collation ⓘ

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

CANCEL

CREATE DATABASE

Com podem apreciar, la nostra base de dades “videoclub” ja es troba al servidor local.

MongoDB Compass - localhost:27017

Connect View Help

Accés a dades

4 DBS 2 COLLECTIONS

filter

> admin

> config

> local

> videoclub

localhost:27017 STANDALONE MongoDB 4.0.4 Enterprise

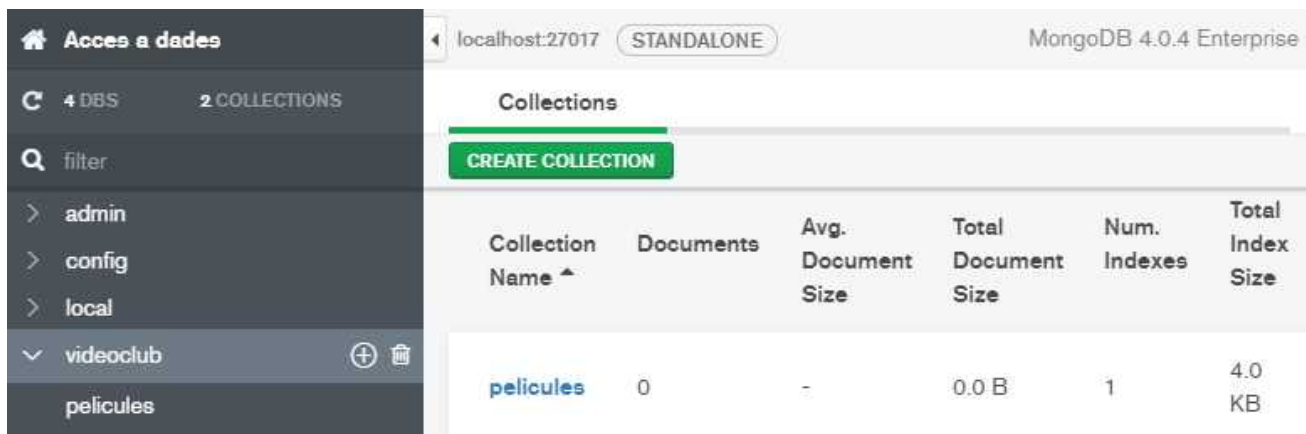
Databases

Performance

CREATE DATABASE

Database Name	Storage Size	Collections	Indexes	
admin	16.0KB	0	1	
config	24.0KB	0	2	
local	36.0KB	1	1	
videoclub	4.0KB	1	1	

Així com també la nostra primera col·lecció “pel·lícules”:



The screenshot shows the MongoDB Compass interface. On the left, the 'Access to data' sidebar lists databases (4 DBS) and collections (2 COLLECTIONS). The 'videoclub' database is expanded, showing the 'peliculas' collection. The main panel displays the 'Collections' tab with a 'CREATE COLLECTION' button. Below this is a table listing the collections.

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
peliculas	0	-	0.0 B	1	4.0 KB

## 5.2 Importació de dades d'un document JSON

Una volta creada la base de dades i la primera col·lecció, cal omplir-la amb dades. N'hi ha varies formes de fer-ho. En este moment anem a utilitzar la ferramenta “**mongoimport**” que s'instal·la junt amb el Compass.

El primer pas serà col·locar en la carpeta on estan els fitxers binaris de MongoDB el fitxer JSON que volem importar. Generalment, a Windows serà la següent carpeta:

```
C:\Program Files\MongoDB\Server\4.0\bin
```

A continuació, haurem d'obrir la consola d'ordres de Windows (CMD) i situar-nos en eixa carpeta. Des d'esta ubicació, caldrà executar la següent ordre:

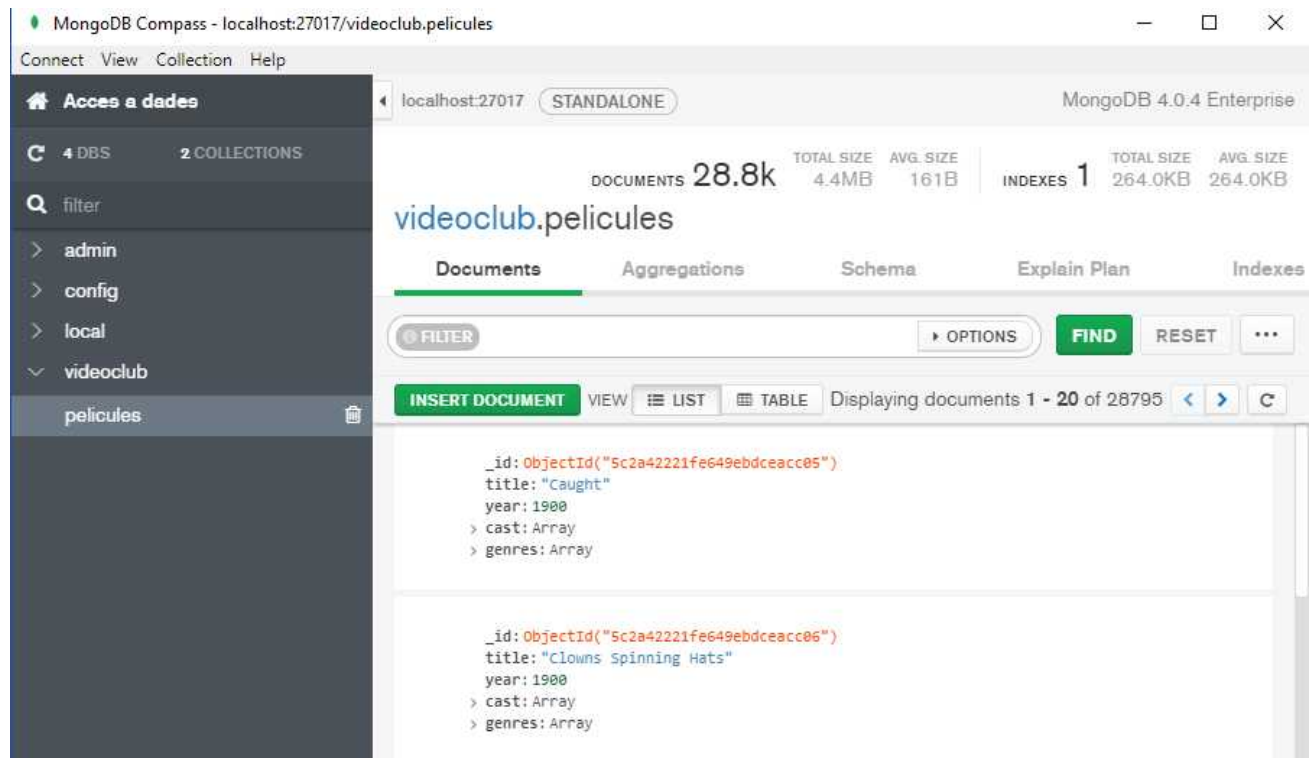
```
C:\Program Files\MongoDB\Server\4.0\bin>mongoimport --jsonArray
--db videoclub --collection peliculas --file movies.json
2018-12-31T17:21:54.285+0100    connected to: localhost
2018-12-31T17:21:54.886+0100    imported 28795 documents
C:\Program Files\MongoDB\Server\4.0\bin>
```

Com podem vore, després d'executar l'ordre “**mongoimport**”, ens indica que s'ha connectat al servidor *localhost*, i que ha importat *28.795 documents*.

Si tornem al MongoDB Compass i refresquem la vista de la col·lecció “*pel·lícules*”, vegem que ja tenim tots els documents carregats. Ací es pot vore una vista de la

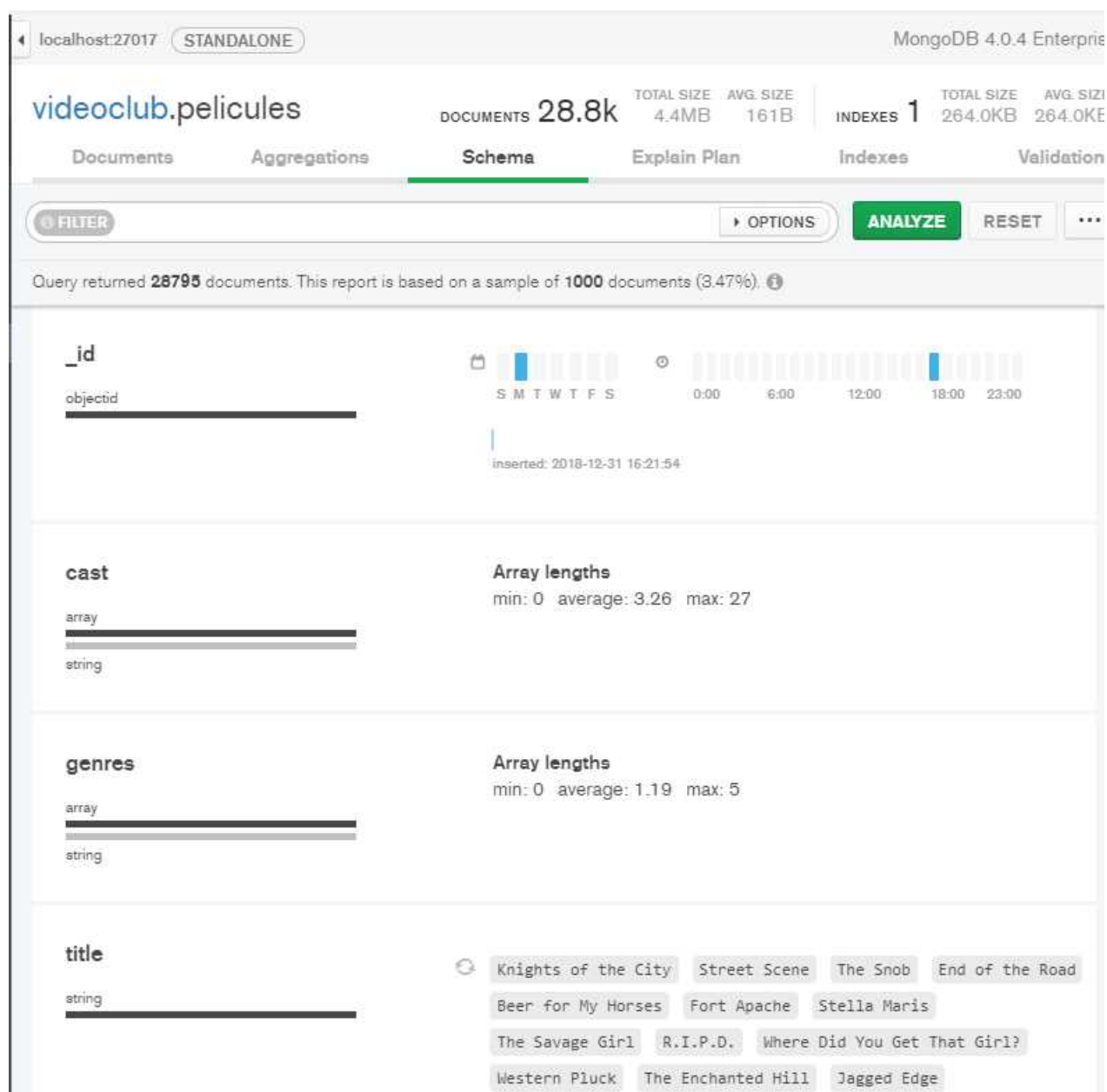
## UD6 - Persistència en BBDD documentals. MongoDB

pestanya “**Documents**” on podem explorar tots els documents que hi ha a la base de dades.



Si ens desplacem a la pestanya “**Schema**” podrem veure un anàlisi de l’estructura dels documents que conformen la nostra base de dades.

Ací podrem veure quins són els tipus de dades de cadascun dels camps, el número d’elements que tenen els vectors de dades (arrays), i moltes altres dades interessants sobre esta col·lecció de dades.



## 5.3 mongoimport

Anem a explicar un poc millor este comando que hem utilitzat en l'apartat anterior. **mongoimport** és capaç d'importar dades en diferents formats a la nostra base de dades. Ara anem a explicar-ho utilitzant un arxiu JSON.

En primer lloc caldria descarregar o generar el nostre fitxer JSON. Una volta tenim el nostre fitxer descarregat, anem a vore com l'importem:

```
mongoimport --db ciudadesdb --collection ciutats --drop --file zips.json --port 8080
```

- Amb el **paràmetre «-db»** li indiquem que volem guardar les dades en la següent base de dades. En este cas «ciudadesdb».
- Amb el **paràmetre «-collection»** li indiquem la col·lecció, és a dir, la taula on volem guardar les dades. En este cas «ciutats».
- El **paràmetre «-drop»** li indica a mongoimport que esborre totes les dades que hi haguera ja en la col·lecció. És a dir, esborraria totes les dades que tinguera la taula abans de realitzar la importació de les dades. Aneu amb compte amb este paràmetre, **si volem afegir les noves dades a les ja existents, no hem d'incloure este paràmetre.**
- El **paràmetre «-file»** és on li indiquem a mongoimport on està el fitxer en format json amb les dades que volem importar.

I açò seria suficient en el cas que el nostre servidor:

1. Estiga en l'equip local, és a dir, en «localhost».
2. El port que està escoltant és el de per defecte de la instal·lació de MongoDB .
3. La base de dades no té nom d'usuari ni contrasenya per a accedir a ella.

### 5.3.1 Paràmetres avançats per a mongoimport.

Com en molts dels casos no es donarà alguna d'estes opcions, haurem d'utilitzar algun dels següents paràmetres per a poder configurar-ho correctament:

- El paràmetre **«-host»** junt amb l'url on es troba el servidor: per a indicar-li a mongoimport en quina direcció està el nostre servidor.
- El paràmetre **«-port»** i el port que hem configurat: per a indicar-li el port que està escoltant el nostre MongoDB.
- Per a autenticar-nos en el servidor haurem d'utilitzar els següents paràmetres:
  - **«-username»** o en versió curta «-u» seguit del nom d'usuari que volem utilitzar: Per a indicar-li el nom d'usuari que hem d'usar.
  - **«-password»** o «-p» seguit de la contrasenya desitjada: per a indicar-li la contrasenya utilitzarem.

Amb estos paràmetres ja podrem indicar el servidor que volem usar, i l'usuari i contrasenya per a validar-nos.





## 6 Comandos bàsics

Sempre que vulguem treballar amb les dades d'una base de dades s'ha de seguir una nomenclatura definida. Primerament apareixeran les lletres **db**, seguides d'un **punt**, i el **nom de la col·lecció** sobre la qual es vol treballar, en este cas, assignatures. Posteriorment ha d'indicar-se el **comando** que es vulga executar, ja siga find() insert()

### 6.1 Insert()

A continuació podeu comprovar que en una mateixa col·lecció es poden inserir diferents documents amb diferents esquemes, utilitzant el comando **insert()**

```
db.collection.insert(...)
```

```
> db.assignatures.insert({nom: 'Química', professor: 'Juan',  
any: '2012'})  
> db.assignatures.insert({nom: 'Física', professor: 'Juan',  
colegi: 'Freelance', edat: '36'})
```

#### Operacions de creació:

- db.collection.insert()
- db.collection.insertOne()
- db.collection.insertMany()

### 6.2 find()

Per a realitzar consultes a la base de dades, haurem d'usar el comando **db.nom\_de\_coleccion.find()**. Este comando pot rebre dos paràmetres: una **consulta** i una **projecció**. Els dos comandos són opcionals pel que podem executar el següent comando sense cap paràmetre:

```
db.assignatures.find()
```

De totes maneres, en executar el comando vorem que el resultat no està massa formatat i és molt difícil llegir-lo. Per a solucionar este problema podem usar el modificador **pretty** que ens retornarà un resultat molt més llegible.

```
db.assignatures.find().pretty( )
```

Ara afegirem la **consulta** al comando find, per a què filtre els elements segons les nostres necessitats. Per a això especificarem un objecte JSON com a primer paràmetre del comando, amb els camps pels que volem filtrar:

```
db.assignatures.find({professor:"Juan"}).pretty( )
```

```
db.assignatures.find({nom:"Quimica", professor:"Juan"}).pretty( )
```

D'esta manera, el comando ens retorna només els camps que volem, a més del \_id. El \_id per defecte es mostra sempre, així que si volem ocultar-ho cal especificar-ho en la projecció.

```
db.assignatures.find({nom: "Quimica", professor:"Juan"}, { professor:1, _id:0, col·legi:1}).pretty( )
```

### Buscar dins d'una matriu (array)

Si volem buscar un sol element dins d'una matriu(array) cal fer una consulta similar a la següent:

```
db.people.find({tags:"laborum"},{name:1,tags:1})
```

## 6.3 Exemples

### Exemple 1

#### **Cree les col·leccions “empleat” i “departament”:**

```
db.createCollection("empleat");  
db.createCollection("departament");
```

#### **Guarde dades en la col·lecció “departament”:**

```
db.createCollection("departament");  
db.departament.insert({_id: 1, departament: "Gerent"});  
db.departament.insert({_id: 2, departament: "Contable"});
```

#### **Guarde dades en la col·lecció “empleat”:**

```
db.empleat.insert(  
  {  
    _id: 1,  
    nom: { primer: 'Javi', cognom: 'Moreno' },  
    ciutat: 'Alcudia',  
    departament: 1  
  }  
);  
  
db.empleat.insert(  
  {  
    _id: 2,  
    name: { primer: 'Esther', cognom: 'Aguado' },  
    ciutat: 'Algemesi',  
    department: 2  
  }  
);
```

## Exemple 2

### Col·lecció autors:

```
db.createCollection("authors");

db.authors.insertMany([
  {
    _id:"a1",
    name: {
      first:"Orlando",
      last:"Becerra" },
    age: 27
  },
  {
    _id:"a2",
    name: {
      first:"mayra",
      last:"sanchez" },
    age: 21
  }
]);
```

### Col·lecció Categories:

```
db.categories.insertMany([
  {
    _id: "c1",
    name: "sci-fi"
  },
  {
    _id: "c2",
    name: "romanç"
  }
]);
```

### Col·lecció Books:

```
db.books.insert([
{
  _id: "b1",
  name: "Groovy Book",
  category: "c1",
  authors: ["a1"]
});
```

```
db.books.insert (
{
  _id: "b2",
  name: "Java Book",
  category: "c2",
  authors:["a1","a2"]
});
```

### Col·lecció Prèstecs:

```
db.lendings.insertMany([
{
  _id: 'l1',
  book: 'b1',
  date: new Date ('01/01/11'),
  lendingBy: 'jose'
},
{
  _id: 'l2',
  book: 'b1',
  date: new Date('02/02/12'),
  lendingBy: 'maria'
}
]);
```

## 7 Pràctica 1

Crea les col·leccions corresponents. Cal afegir **relacions** entre **productes-botiga** (UN producte es ven en UNA botiga) i entre **producte-client** (UN client compra MOLTS productes).

### Col·lecció Clients:

```
{
  "_id": "1",
  "nom": "Pedro Ramírez",
  "telèfons": [
    "465465456",
    "4545654"
  ],
  "direccio": {
    "carrer": "Valencia",
    "numero": 25,
    "extra": "Portal 7, 4t esquerra",
    "ciutat": "Xativa"
  }
}
{
  "_id": "2",
  "nom": "Ramón Ramírez",
  "telèfons": [
    "465465456"
  ],
  "direccio": {
    "carrer": "Alzira",
    "numero": 26,
    "extra": "Portal 8, 4t esquerra",
    "ciutat": "Xativa"
  }
}
```

### Col·lecció Productes:

```
{  _id: 860
  nom: "Portàtil Asus",
  quantitat: 25,
  preu: 459.99
},
{  _id: 870
  nom: "Portàtil HP",
  quantitat: 1,
  preu: 765.50
},
{
  _id: 890,
  nom: "Portàtil Lenovo",
  quantitat: 7,
  preu: 800
},
{  _id: 900
  nom: "HDD Seagate",
  quantitat: 45,
  preu: 79.99,
  tipus: "HDD"
},
{  _id: 910
  nom: "HDD Maxtor",
  quantitat: 20,
  preu: 65.50,
  tipus: "HDD"
}
```



**Col·lecció Botigues:**

```
{
  _id: ObjectId("XXXX"),
  nom: 'Gran Plaza'
},
{
  _id: ObjectId("YYYY"),
  nom: 'Dues Germanes',
  central: ObjectId("XXXX")
},
{
  _id: ObjectId("ZZZZ"),
  nom: 'Carles V',
  central: ObjectId("XXXX")
},
{
  _id: ObjectId("HHHH"),
  nom: 'Avd Ciències',
  central: [ObjectId("XXXX"), ObjectId("ZZZZ")]
},
{
  _id: ObjectId("AAAA"),
  nom: 'Caputxins',
  servicis: {
    electrònica: 'Abart',
    consumibles: 'Sisco',
    comandes: ObjectId("XXXX")
  }
}
```

Hem utilitzat el document ***servicis*** com a part del document ***botiga***.

El document *servicis* conté els proveïdors associats a la tipologia de productes que poden trobar-se a la botiga, com poden ser la gamma de productes frescos, congelats, dietètics, o d'oci.

```
>db.botiga.find({central: ObjectId("Gran Plaza")});
```

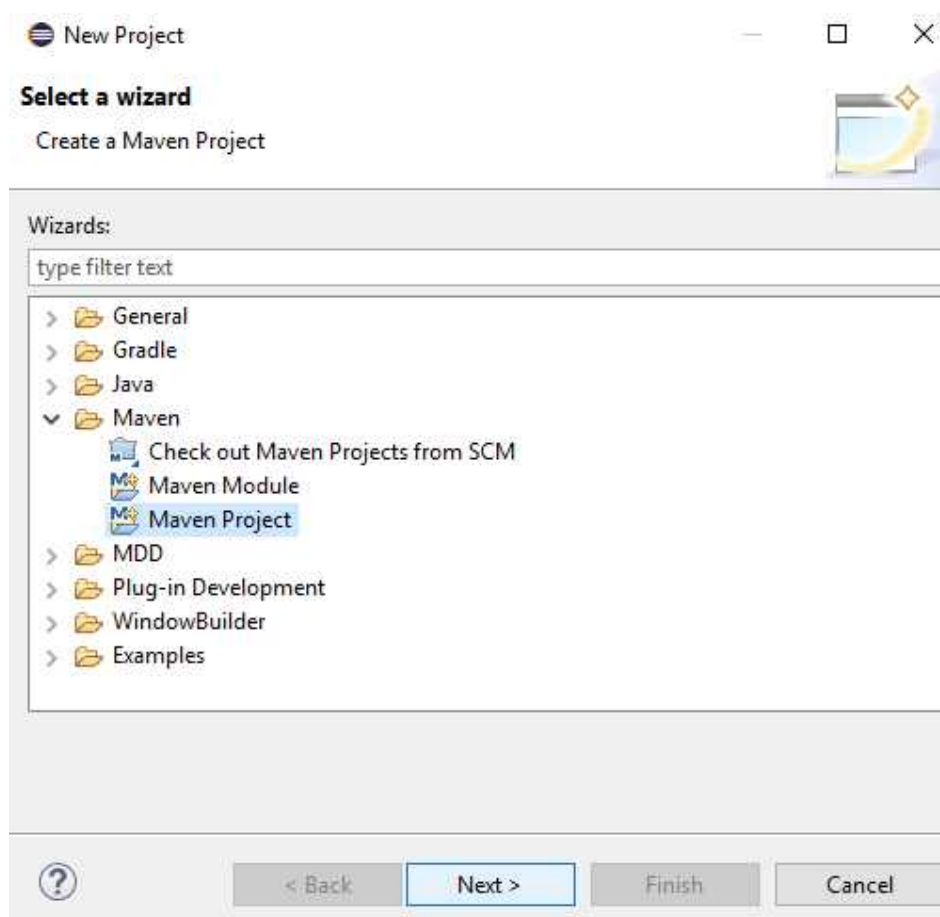
## 8 Connexió a MongoDB desde Java

### 8.1 Creació i configuració projecte Java amb connexió amb MongoDB

La forma recomanada de començar a utilitzar un dels drivers del projecte és amb un sistema de gestió de dependències. En el nostre cas, utilitzarem el gestor vist en la unitat 3: Maven.

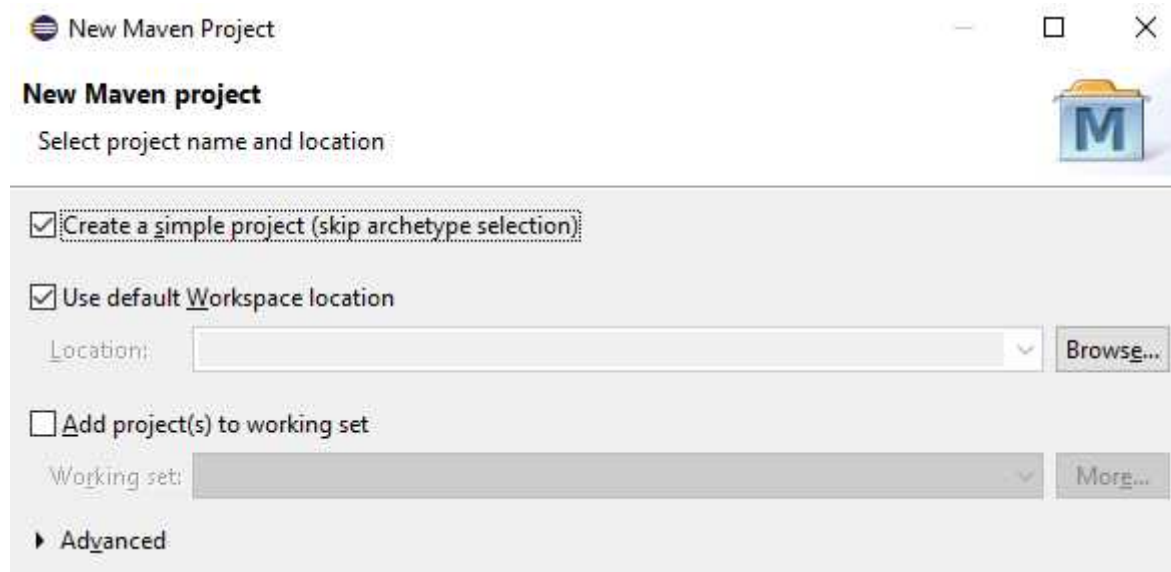
Per començar hem de crear un projecte Maven:

#### 1.- File → New Project → Maven → Maven Project



2.- Marquem les opcions que apareixen a la següent captura.

## UD6 - Persistència en BBDD documentals. MongoDB



New Maven Project

**New Maven project**

Select project name and location

☒ Create a simple project (skip archetype selection)

☒ Use default Workspace location

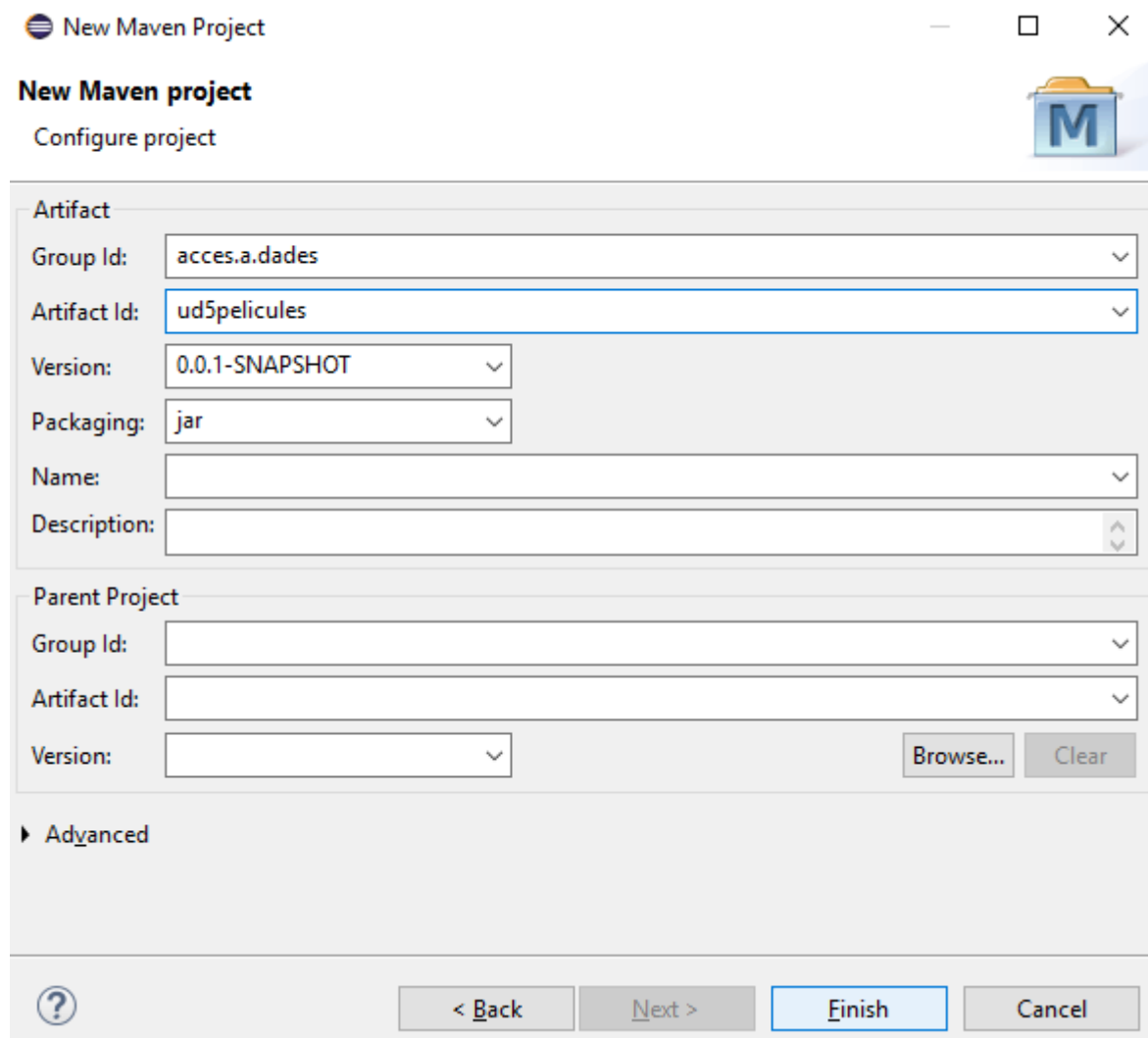
Location:  Browse...

☐ Add project(s) to working set

Working set:  More...

▶ Advanced

3.- Posem les següents dades en el formulari i polsem "Finish".



New Maven Project

**New Maven project**

Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:  Browse... Clear

▶ Advanced

? < Back Next > Finish Cancel

#### 4.- Ara cal configurar el fitxer de configuració de Maven, “**pom.xml**”

Per a fer-ho, hem d’afegir el següent codi XML al ja existent:

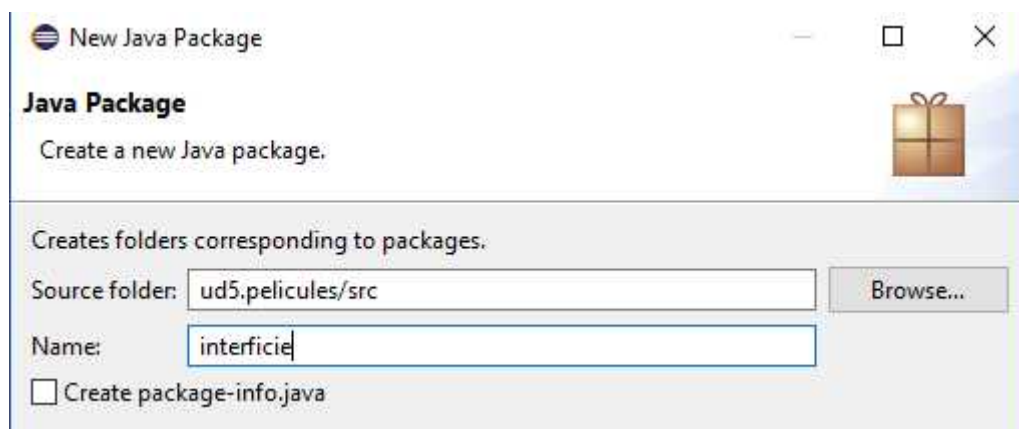
```
<dependencies>
  <dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>3.12.7</version>
  </dependency>
</dependencies>
```

Així, el fitxer **pom.xml** quedarà de la següent manera:

```
pom.xml
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>accés.a.dades</groupId>
4   <artifactId>ud5.pelicules</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6   <dependencies>
7     <dependency>
8       <groupId>org.mongodb</groupId>
9       <artifactId>mongo-java-driver</artifactId>
10      <version>3.12.7</version>
11    </dependency>
12  </dependencies>
13 </project>
```

## 8.2 Creem la classe “*ObtindrePeliclesMongoDB*”

1.- Dins la carpeta “src”, creem un paquet “interficie”.



2.- En este paquet creem una classe pública **ObtindrePeliclesMongoDB.java** que tinga un mètode **main**.

New Java Class

Java Class

Create a new Java class.

Source folder: ud5.pelicules/src/main/java Browse...

Package: interficie Browse...

☐ Enclosing type: Browse...

Name: ObtindrePeliclesMongoDB

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

☒ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Finish Cancel

El resultat serà el següent:

```

1 package interficie;
2
3 public class ObtindrePeliclesMongoDB {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7
8     }
9
10 }
11

```

### 8.2.1 Llibreries de classes per a importar

Per poder utilitzar totes les funcionalitats del controlador de MongoDB, cal importar les següents llibreries:

```

package interficie;

//NOVA API MongoClient (des de la versió 3.7)
import com.mongodb.ConnectionString;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoClient;

//ELEMENTS COMUNS A TOTES LES API'S de MongoClient

import com.mongodb.ServerAddress;

import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;

import org.bson.Document;
import java.util.Arrays;
import com.mongodb.Block;

import com.mongodb.client.MongoCursor;
import static com.mongodb.client.model.Filters.*;
import com.mongodb.client.result.DeleteResult;
import static com.mongodb.client.model.Updates.*;
import com.mongodb.client.result.UpdateResult;
import java.util.ArrayList;
import java.util.List;

```

## 8.3 Establint una connexió

Cal utilitzar ***MongoClients.create()*** per a establir una connexió amb una instància de MongoDB en execució.

La instància ***MongoClients*** representa un conjunt de connexions a la base de dades; només es necessitarà una instància de la classe ***MongoClient*** fins i tot amb múltiples subprocesos.

### IMPORTANT:

En general, només es crea una instància de MongoClient per a una implementació de MongoDB determinada (per exemple, standalone, replica set o un clúster fragmentat) i s'utilitza en tota l'aplicació.

No obstant això, si es necessitaren múltiples instàncies:

- Tots els límits d'ús de recursos (per exemple, connexions màximes, etc.) s'apliquen per instància de MongoClient.
- Per a eliminar una instància, cal cridar a MongoClient.close() per a netejar els recursos.

### 8.3.1 Connectar-se a una única instància de MongoDB

Existixen diverses maneres de connectar-se a un únic servidor MongoDB.

1. Es pot instanciar un objecte MongoClient **sense cap paràmetre** per a connectar-se a una instància MongoDB que s'execute en localhost en el port 27017:

```
public static void main(String[] args) {  
    MongoClient mongoClient = MongoClient.create();  
}
```

2. Es pot especificar explícitament el **nom d'host** que es connectarà a una instància de MongoDB que s'execute en l'host especificat en el **port 27017**:

```
public static void main(String[] args) {  
    MongoClient mongoClient = MongoClient.create(  
        MongoClientSettings.builder()  
            .applyToClusterSettings(builder ->  
                builder.hosts(Arrays.asList(new ServerAddress("localhost"))))  
            .build());  
}
```

3. Es pot especificar explícitament el **nom d'host i el port**:

```
public static void main(String[] args) {  
    MongoClient mongoClient = MongoClient.create(  
        MongoClientSettings.builder()  
            .applyToClusterSettings(builder ->  
                builder.hosts(Arrays.asList(new ServerAddress("localhost", 27018))))  
            .build());  
}
```

4. Es pot especificar la **ConnectionString**:

```
public static void main(String[] args) {  
    MongoClient mongoClient = MongoClient.create("mongodb://localhost:27017");  
}
```

La cadena de connexió segueix principalment el **RFC 3986**, amb l'excepció del nom de domini. Per a MongoDB, és possible llistar múltiples noms de domini separats per una coma.

## 8.4 Obtenint una Base de dades

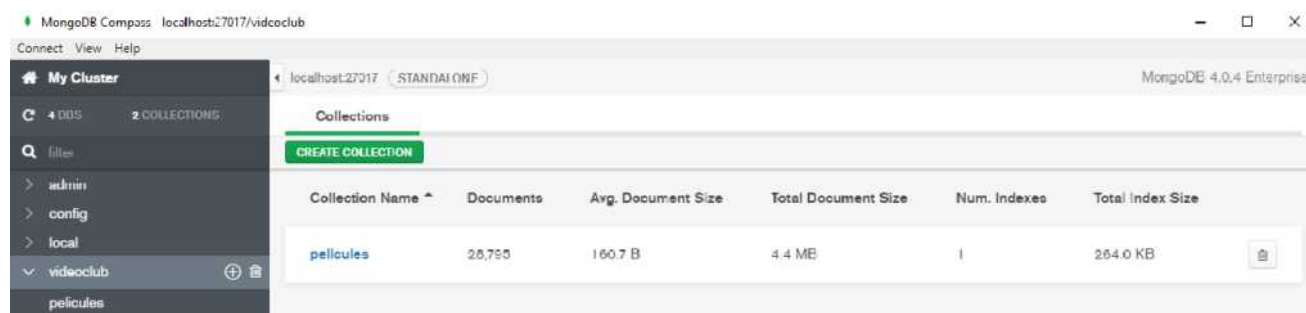
Una volta tenim una instància de MongoClient connectada a la base de dades MongoDB, s'ha d'utilitzar el mètode **MongoClient.getDatabase()**.

Si no existeix una base de dades, MongoDB crea la base de dades quan emmagatzemem per primera vegada les dades per a eixa base de dades.

Anem a recordar quina era la base de dades que hem creat al nostre servidor MongoDB:



## UD6 - Persistència en BBDD documentals. MongoDB



Al servidor hem creat la base de dades “videoclub”. Per obtindre-la des de la nostra aplicació, utilitzarem el següent codi:

```
public static void main(String[] args) {  
    MongoClient mongoClient = MongoClient.create("mongodb://localhost:27017");  
    MongoDBDatabase database = mongoClient.getDatabase("videoclub");  
}
```

Les instàncies de **MongoDatabase** són immutables.

### 8.5 Obtenint una Col·lecció

Ara ja tenim una MongoDBDatabase, i podem accedir a les seues Col·leccions amb el mètode **getCollection()**.

Cal especificar el nom de la col·lecció al cridar al mètode **getCollection()**. Si no existeix la col·lecció, MongoDB la crea quan emmagatzemem dades en ella per primera volta.

```
public static void main(String[] args) {  
    MongoClient mongoClient = MongoClient.create("mongodb://localhost:27017");  
    MongoDBDatabase basedades = mongoClient.getDatabase("videoclub");  
    MongoCollection coleccio = basedades.getCollection("pelicules");  
}
```

## 9 Consultes a la base de dades MongoDB

Per a consultar una col·lecció, es pot utilitzar el mètode **find()** de la col·lecció. Podem cridar al mètode sense cap argument per a consultar tots els documents d'una col·lecció o passar un filtre per a buscar documents que coincideixen amb els criteris de filtre.

El mètode **find()** retorna una instància **FindIterable()** que proporciona una interfície fluida per a encadenar altres mètodes.

### 9.1 Classe Document

En la interacció amb MongoDB usant el driver de Java se sol utilitzar per a quasi tot la classe **Document**. Utilitzarem esta classe no sols per als documents que hàgem d'introduir, sinó per a definir els criteris d'eliminació, actualització, ordenació i altres.

El **constructor** de la classe Document admet dos paràmetres: una clau i un valor. Per a crear documents amb més d'una parella clau-valor aplicarem el mètode **append()** al constructor tantes vegades com siga necessari encadenant les crides amb un punt (.). Esta serà la tècnica bàsica per a construir les llistes JSON.

#### Exemple 1 des de Java:

```
List<String> Hobbies = new ArrayList <String>();

Hobbies.add("musica");
Hobbies.add("pelicules");
Hobbies.add("senderisme");

Document document = new Document ("nom", "Javi").
    append("email", "javi@isca.es").
    append("twitter", "iscajavi").
    append("entreteniments", Hobbies).
    append("location", new Document ("ciutat", "alberic").
        append("zip", 46260));
```

#### Exemple 2: Obtindre valors per al document des d'un Pojo:

```
Document documentNou = new Document ()
    . append("titol", llibre.getTitulo())
    . append("descripcio", llibre.getDescripcio())
    . append("autor", llibre.getAutor())
    . append("data", llibre.getData())
    . append("disponible", llibre.getDisponible());
```

## 9.2 Obtindre el primer document d'una col·lecció

Per a obtenir el primer document de la col·lecció, utilitzem el mètode **find()** sense cap paràmetre i li encadenem el mètode **first()**.

Si la col·lecció està buida, l'operació retorna *null*.

```
public static void main(String[] args) {  
    MongoClient mongoClient = MongoClient.create("mongodb://localhost:27017");  
    MongoDBDatabase basedades = mongoClient.getDatabase("videoclub");  
    MongoCollection<Document> coleccio = basedades.getCollection("pelicules");  
    Document primeraPelícula = coleccio.find().first();  
    System.out.println(primeraPelícula.toJson());  
}
```

Este exemple imprimeix per consola el següent document:

```
{ "_id" : { "$oid" : "5c2a42221fe649ebdceacc05" },  
  "title" : "Caught",  
  "year" : 1900,  
  "cast" : [],  
  "genres" : [] }
```

Com podem apreciar, a més de la informació pròpia del document, també apareix l'element **\_id** que afegix MongoDB automàticament per fer referència interna a eixe document.

### **NOTA:**

MongoDB reserva els noms de camp que comencen amb "\_" i "\$" per a ús intern.

### 9.3 Obtindre tots els documents d'una col·lecció

Per a obtenir tots els documents de la col·lecció, utilitzarem el mètode **find()** sense cap paràmetre.

Per a recórrer els resultats, hem d'encadenar el mètode **iterator()** al **find()**.

En el següent exemple recuperem tots els documents de la col·lecció i imprimim els documents retornats (28.795 documents):

```
MongoCursor<Document> cursor = coleccio.find().iterator();
try {
    while (cursor.hasNext()) {
        System.out.println(cursor.next().toJson());
    }
} finally {
    cursor.close();
}
```

Encara que es permet el següent codi per a la iteració, **CAL EVITAR** el seu ús ja que l'aplicació pot perdre un cursor si el bucle acaba abans d'hora:

```
for (Document cur : coleccio.find()) {
    System.out.println(cur.toJson());
}
```

## 9.4 Classe Filters: especificar un filtre de consulta

Per a buscar documents que complisquen certes condicions, passada un objecte de filtre al mètode find(). Per a facilitar la creació d'objectes de filtre, el controlador Java proporciona l'assistent de [Filters](#).

### Comparació

PREDICAT	DESCRIPCIÓ
<b>eq</b>	Igual
<b>gt</b>	Mayor que
<b>gte</b>	Mayor o igual
<b>lt</b>	Menor
<b>lte</b>	Menor o igual
<b>ne</b>	No es igual
<b>in</b>	Busca algún valor
<b>nin</b>	Ningún dels valors de una matriu (array)

### Matrius (arrays)

OPERADOR	DESCRIPCIÓ
<b>all</b>	Busca matrius (arrays) que contenen els elements especificats en la consulta.
<b>elemMatch</b>	Selecciona tots els documents si un element de la matriu (array) coincideix amb tot el que hem especificat en \$elemMatch.
<b>size</b>	Selecciona tots els documents si el camp de la matriu (array) es un tamaño específico.

## Lògics

OPERADOR	DESCRIPCIÓ
<b>and</b>	I
<b>or</b>	O
<b>not</b>	No
<b>nor</b>	Selecciona tots els documents que no complixen les condicions especificades.

## Exemples

- Buscar els documents en els que el país es “finlandia”:

```
Bson filter =Filters.eq("pais","finlandia");
```

- **AND** Busca per la condició logo = "" y poblacio="Helsinki”:

```
Bson filter =Filters.and(
    Filters.eq("logo",""), Filters.eq("poblacio","Helsinki")
); //AND
```

- **OR** Busca documents amb logo ="" o codi=8579:

```
Bson filter =Filters.or(
    Filters.eq("logo",""),
    Filters.eq("codi",8579)
); //OR
```

- Busca Codi=8579 o Població="Espoo" o País = "Espanya" o Sigles = "fi"

```
Bson filter =Filters.and(
    Filters.or(
        Filters.eq("Codi", 8579),
        Filters.eq("Poblacio","Espoo"),
        Filters.or(
            Filters.eq("Pais", "Espanya"),
            Filters.eq("Sigles", "fi")
        ) //or
    ) //or
); //and
```

## Exemples matrius (arrays)

- Selecciona tots els documents amb una matriu (array) “Provincies” que continga la província “Alacant”:

```
Bson filter = Filters.all("Provincies", "Alacant");
```

- Selecciona tots els documents amb una matriu (array) “Provincies” que continga les províncies “Alacant” y “Valencia”:

```
Bson filter = Filters.all("Provincies",
    Arrays.asList("Valencia", "Alacant")
);
```

### 9.4.1 Obtindre un document únic que coincidisca amb un filtre

Per exemple, per a trobar el primer document on el camp “year” té el valor “2017”, li passem un objecte de filtre **eq** per a especificar la condició d'igualtat:

```
package interficie;

import org.bson.Document;
import org.bson.conversions.Bson;

import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import static com.mongodb.client.model.Filters.*;

public class ConsultaPelicles {

    public static void main(String[] args) {

        MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017");

        MongoDatabase basedades = mongoClient.getDatabase("videoclub");

        MongoCollection<Document> coleccio = basedades.getCollection("pelicles");

        //5.1 Obtindre 1 pel·lícula de l'any 2017
        Document pel·lícula2017 = coleccio.find(eq("year", 2017)).first();

        System.out.println(pel·lícula2017.toJson());

    }
}
```

Este codi ens imprimirà el següent document:

```
{ "_id" : { "$oid" : "5c2a42221fe649ebdceb3ac2" },  
  "title" : "Monster Trucks",  
  "year" : 2017,  
  "cast" : ["Lucas Till", "Jane Levy", "Amy Ryan", "Holt McCallany", "Rob Lowe",  
  "Danny Glover", "Frank Whaley", "Chad Willett", "Barry Pepper", "Thomas Lennon", "Tucker Albrizzi"],  
  "genres" : ["Animated", "Adventure", "Science Fiction"] }
```

## 9.4.2 Obtindre tots els documents que coincideixen amb un filtre

### Exemple 1:

El següent exemple retorna i imprimeix totes les pel·lícules que s'han llançat al mercat després de 2017:

```
//5.3.2 Obtindre totes les pel·lícules des de 2017 fins ara  
Block<Document> printBlock = new Block<Document>() {  
    @Override  
    public void apply(final Document document) {  
        System.out.println(document.toJson());  
    }  
};  
  
coleccio.find(gt("year", 2017)).forEach(printBlock);
```

L'exemple utilitza el mètode *forEach* en l'objecte *FindIterable* per a aplicar un bloc a cada document.

### EXERCICI 0A:

Com es pot apreciar, el `forEach()` apareix amb una ratlla damunt. Açò vol dir que este mètode està “*deprecated*” és a dir que en pròximes versions desapareixerà. Implementa un bloc de codi que faci el mateix, però sense utilitzar el `forEach()`.

Per a especificar un filtre per a un rang de valors podem utilitzar el *helper* `and` i el `or`, entre d'altres.

### Exemple 2:

El següent exemple retorna i imprimeix totes les pel·lícules que s'han llançat entre els anys 1998 i 2000:



```
// 5.3.2 Obtindre totes les pel·lícules entre els anys 1998 i 2000
Block<Document> printBlock = new Block<Document>() {
    @Override
    public void apply(final Document document) {
        System.out.println(document.toJson());
    }
};

coleccio.find(and(gt("year", 1998), (lt("year", 2000)))).forEach(printBlock)
```

### EXERCICI 0B:

Implementa un bloc de codi que faci el mateix, però sense utilitzar el `forEach()`.

## 9.5 Contar documents en una col·lecció

Per a contar el nombre de documents d'una col·lecció, es pot utilitzar el mètode **`countDocuments()`** de la col·lecció.

El següent codi ha d'imprimir 28.975, que són els documents que té la col·lecció “pel·lícules”.

```
// 5.4 Contar els documents que té la col·lecció "pel·lícules"
System.out.println("La col·lecció \"pel·lícules\" té "+coleccio.countDocuments());
```

## 9.6 Buscar patrons en els camps dels documents

Per buscar una expressió regular en un camp, s'utilitza l'operador **regex**. Per crear els patrons, a més del text, podem utilitzar caràcters comodí:

**^Mary \*** → Vol dir que comence per “Mary ” i després que tinga els caràcters que vullga.

**.** → Vol dir un caràcter.

**\*** → Vol dir cap o molts caràcters.

**\* Poppins\$** → Vol dir que el text acaba en “ Poppins”.

### Exemple 3:

Extrau totes les pel·lícules que tenen el text “Mary Poppins” en el camp “title”:

```
//5.5 Buscar el patró "Mary Poppins" en el camp "title" dels documents
Block<Document> printBlock = new Block<Document>() {
    @Override
    public void apply(final Document document) {
        System.out.println(document.toJson());
    }
};

coleccio.find(regex("title", "Mary Poppins")).forEach(printBlock);
```

#### Exemple 4:

Busca totes les pel·lícules que comencen per “Mary ” en el camp “title” i imprimix quantes hi ha. Este exemple ha de tornar 16 pel·lícules.

```
//5.5.2 Buscar les pel·lícules que comencen per "Mary " en el camp "title"
//i dir quantes hi ha

long numDocuments = coleccio.countDocuments(Filters.regex("title", "^Mary *"));
System.out.println("Ha trobat "+numDocuments+" pel·lícules.");
```

#### Exemple 5:

Extrau totes les pel·lícules que acaben en “ Poppins” en el camp “title”:

```
//5.5.3 Buscar les pel·lícules que acaben per " Poppins" en el camp "title"
Block<Document> printBlock = new Block<Document>() {
    @Override
    public void apply(final Document document) {
        System.out.println(document.toJson());
    }
};

coleccio.find(regex("title", " Poppins$")).forEach(printBlock);
```

## 9.7 Consultes sobre arrays

### 9.7.1 Buscar elements dins d'un array

Per buscar si un element es troba dins d'un camp de tipus Array, utilitzarem l'operador \$all.

### Exemple 6:

Extrau totes les pel·lícules en les que ha participat l'actor "Danny Glover":

```
// 5.6.1 Buscar les pel·lícules que en les que ha participat l'actor "Danny Glover"
Block<Document> printBlock = new Block<Document>() {
    @Override
    public void apply(final Document document) {
        System.out.println(document.toJson());
    }
};

coleccio.find(all("cast", "Danny Glover")).forEach(printBlock);
```

## 9.7.2 Contar els elements que conté un array

Per saber el número d'elements que té un camp de tipus array, utilitzarem l'operador \$size.

### Exemple 7:

Extrau totes les pel·lícules en les que han participat 6 actors:

```
// 5.6.2 Buscar les pel·lícules que en les que han participat 6 actors
Block<Document> printBlock = new Block<Document>() {
    @Override
    public void apply(final Document document) {
        System.out.println(document.toJson());
    }
};

coleccio.find(size("cast", 6)).forEach(printBlock);
```

## 9.8 Projeccions

Al realitzar una consulta i imprimir tot el document, ens apareixen tots els camps que té cada document. Tal volta únicament estem interessats en algun d'estos camps.

Quan volem que sols apareguen una part dels camps dels documents resultants d'una consulta, el que necessitem és fer una **projecció** dels resultat.

En MongoDB açò es pot fer de dues maneres:

### 1. Especificant literalment com volem que siga el document resultant:

```
//5.7.1 Mostra el títol i els actors de totes les comèdies de l'any 2000
Block<Document> printBlock = new Block<Document>() {
    @Override
    public void apply(final Document document) {
        System.out.println(document.toJson());
    }
};

coleccio.find(and(eq("year", 2000), all("genres", "Comedy")))
    .projection(new Document("title", 1)
        .append("cast", 1)
        .append("_id", 0))
    .forEach(printBlock);
```

### 2. Utilitzant la classe **Projections** que proporciona el driver de MongoDB. D'esta manera indicarem quins camps volem que apareguen en els documents resultants de la consulta d'una manera més intuïtiva.

```
// 5.7.2 Mostra el títol i els actors de totes les comèdies de l'any 2000
Block<Document> printBlock = new Block<Document>() {
    @Override
    public void apply(final Document document) {
        System.out.println(document.toJson());
    }
};

coleccio.find(and(eq("year", 2000), all("genres", "Comedy")))
    .projection(fields(include("title", "cast"), excludeId()))
    .forEach(printBlock);
```

## 9.9 Ordenant els resultats

Si volem ordenar documents, cal passar-li un document d'especificació d'ordenació al mètode **FindIterable.sort()**. El controlador Java proporciona **Sorts helpers**, per facilitar el document d'especificació de la classificació.

### Exemple 8:

Extrau el títol i els actors de totes les comèdies de l'any 2008 ordenades alfabèticament:

```
// 5.8.1 Mostra el títol i els actors de totes les comèdies de l'any 2008
// ordenades alfabèticament
Block<Document> printBlock = new Block<Document>() {
    @Override
    public void apply(final Document document) {
        System.out.println(document.toJson());
    }
};

coleccio.find(and(eq("year", 2008), all("genres", "Comedy")))
    .sort(Sorts.ascending("title"))
    .projection(fields(include("title", "cast"), excludeId())).forEach.forEach(printBlock);
```

## 10 Inserció de documents a la base de dades MongoDB

Per a este apartat, anem a crear una base de dades de prova anomenada “*software*” i dins d’ella, una col·lecció “*aplicacions*”.

```
public static void main(String[] args) {  
    MongoClient mongoClient = MongoClient.create("mongodb://localhost:27017");  
    MongoDBDatabase basedades = mongoClient.getDatabase("software");  
    MongoCollection<Document> coleccio = basedades.getCollection("aplicacions");  
}
```

### 10.1 Crear un document

Per a crear el document utilitzant el controlador Java, utilitzem la classe **Document**. Per exemple, considerem el següent document de JSON:

```
{  
  "nom": "MongoDB",  
  "tipus": "base de dades",  
  "conte": 1,  
  "versions": "v3.2", "v3.0", "v2.6" ],  
  "info": { x : 203, i : 102 }  
}
```

Per a crear el document utilitzant el controlador Java, s’ha d’instanciar un objecte **Document** amb un camp i un valor, i utilitzar el seu mètode **append()** per a incloure camps i valors addicionals a l’objecte document. El valor pot ser un altre objecte Document per a especificar un document incrustat:

```
//6.1 Crear Documents  
Document doc = new Document("nom", "MongoDB")  
    .append("tipus", "database")  
    .append("conte", 1)  
    .append("versions", Arrays.asList("v3.2", "v3.0", "v2.6"))  
    .append("info", new Document("x", 203).append("y", 102));
```

#### **NOTA:**

*El tipus d’array BSON correspon al tipus Java `java.util.List`.*

## 10.2 Inserir un Document

Una vegada que tenim l'objecte **MongoCollection**, es pot inserir documents en la col·lecció. Per inserir un sol document en la col·lecció, es pot utilitzar el mètode **insertOne()** de classe **Collection**.

```
//6.2.1 Inserir un document
coleccio.insertOne(doc);
```

### NOTA:

Tal i com hem vist a l'apartat 5.1, si no s'especifica cap camp **\_id** de nivell superior en el document, MongoDB afegirà automàticament el camp **\_id** al document inserit.

## 10.3 Inserir Múltiples Documents

Per a afegir múltiples documents, es pot utilitzar el mètode **insertMany()** de la classe **Collection** que pren una llista de documents per a inserir.

El següent exemple agregarà múltiples documents de la forma:

```
{ "contador" : valor }
```

Creem els documents en un bucle i els afegim a una llista de documents:

```
//6.2.2 Inserir molts documents
List<Document> documents = new ArrayList<Document>();
for (int valor = 0; valor < 100; valor++) {
    documents.add(new Document("contador", valor));
}
```

Per inserir estos documents en la col·lecció, li passem la llista de documents al mètode **insertMany()**:

```
coleccio.insertMany(documents);
```

Una volta inserits, podem comprovar que en la nova col·lecció que hem creat a la base de dades, s'han creat realment. Una forma de vore-ho ràpidament seria consultant el número de documents que té la col·lecció.

```
System.out.println("La col.lecció té "+coleccio.countDocuments()+" documents.");
```

## 11 Actualització de documents a la base de dades MongoDB

Per a este apartat, anem a continuar amb la base de dades de prova creada en l'apartat anterior.

Si volem actualitzar els documents d'una col·lecció, es poden utilitzar els mètodes **updateOne** i **updateMany** de classe **Collection**.

Cal passar als mètodes:

- Un **objecte de filtre** per a determinar el document o els documents a actualitzar. Per a facilitar la creació d'objectes de filtre, el controlador Java proporciona el **Filters** helper. Per a especificar un filtre buit (és a dir, que coincidisca amb tots els documents d'una col·lecció), utilitzarem un objecte **Document** buit.
- Un **document d'actualització** que especifica les modificacions. Per a obtindre una llista dels operadors disponibles, consulte [Actualitzar operadors](#).

Els mètodes d'actualització retornen un **UpdateResult** que proporciona informació sobre l'operació, incloent el nombre de documents modificats per l'actualització.

### 11.1 Actualitzar un únic document

Per tal d'actualitzar com a molt un document, utilitzem el mètode **updateOne**.

En el següent exemple s'actualitza el primer document que complisca la condició del filtre “contador” és igual a **10** i li canvia el valor de “contador” a **110**.

```
MongoClient mongoClient = MongoClient.create("mongodb://localhost:27017");
MongoDatabase basedades = mongoClient.getDatabase("software");
MongoCollection<Document> coleccio = basedades.getCollection("aplicacions");
// 7.1 UpdateOne: Modifica el valor del camp "contador" en aquell document que valga 10
coleccio.updateOne(eq("contador", 10), new Document("$set", new Document("contador", 110)));
```



## 11.2 Actualitzar múltiples documents

Per tal d'actualitzar tots els documents que complisquen la condició del filtre, utilitzem el mètode **updateMany**.

En el següent exemple s'incrementa el valor de "contador" en 50, en tots els documents on "contador" és menor que **80**.

```
//7.2 UpdateMany: Incrementa el valor de "contador" en 50,  
//en tots els documents on "contador" és menor que 80  
UpdateResult updateResult = coleccio.updateMany(lt("contador", 80), inc("contador", 50));  
System.out.println(updateResult.getModifiedCount());
```

## 12 Eliminació de documents a la base de dades MongoDB

Continuem amb la base de dades "software" creada en l'apartat 6.

Per tal de borrar documents d'una col·lecció, es poden utilitzar els mètodes **deleteOne** i **deleteMany** de classe **Collection**.

Cal passar als mètodes un objecte de filtre per a determinar el document o documents a esborrar. Per a especificar un filtre buit (és a dir, que coincidisca amb tots els documents d'una col·lecció), utilitzem un objecte **Document** buit.

Els mètodes d'eliminació retornen un **DeleteResult** que proporciona informació sobre l'operació, incloent el nombre de documents esborrats.

### 12.1 Eliminar un únic document que compleix una condició

Per tal d'eliminar com a molt un document que complisca la condició del filtre, utilitzem el mètode **deleteOne**.

En el següent exemple s'esborra el primer document que complisca la condició del filtre "contador" és igual a **110**.

```
// 8.1 DeleteOne: Elimina el document que tinga 110 en el camp "contador".  
coleccio.deleteOne(eq("contador", 110));
```

## *12.2 Eliminar tots els documents que compleixen una condició*

Per tal d'esborrar tots els documents que complisquen la condició del filtre, utilitzem el mètode **deleteMany**.

En el següent exemple s'esborren tots els documents on el valor de "contador" es major o igual a 50.

```
// 8.2 DeleteMany: Elimina tots els documents on "contador" és major o igual a 50
DeleteResult deleteResult = coleccio.deleteMany(gte("contador", 50));
System.out.println(deleteResult.getDeletedCount());
```