

# Apunts CE\_5074 1.1

Iloc: [Institut d'Ensenyaments a Distància de les Illes Balears](#)  
Curs: Sistemes de Big Data  
Llibre: Apunts CE\_5074 1.1

Imprès per: David Ramirez Ruiz  
Data: dissabte, 26 d'octubre 2024, 11:49

## Taula de continguts

### 1. Introducció

### 2. Què és Big Data?

- 2.1. Les dades: una mica d'història
- 2.2. Les dades en l'era digital
- 2.3. Definició de Big Data: les tres V (o més)

### 3. Sistemes de Big Data

### 4. Bases de dades NoSQL

- 4.1. Model clau-valor
- 4.2. Model basat en documents
- 4.3. Model columnar o wide-column
- 4.4. Model basat en grafs

### 5. MongoDB

- 5.1. Creació del clúster
- 5.2. Primeres passes
- 5.3. Interactuar amb les col·leccions
- 5.4. Operacions CRUD
- 5.5. Operadors

## 1. Introducció

Al principi d'aquest lliurament veurem què és el que entenem per *big data* o dades massives. Veurem alguns exemples que mostren algunes de les seves principals característiques i analitzarem la definició més àmpliament utilitzada, la de les 3 V: volum, varietat i velocitat.

A continuació introduirem els sistemes de big data, els quals s'estructuren en diverses capes que es corresponen a les diferents etapes del procés d'extracció d'informació rellevant a partir de les dades.

La resta del lliurament la dedicarem a una d'aquestes capes, la d'emmagatzematge (amb la qual continuarem en lliuraments posteriors), i ens centrarem en les bases de dades NoSQL. Estudiarem els diferents models de bases de dades NoSQL, i per acabar, introduirem la més utilitzades avui en dia, MongoDB, una base de dades basada en documents. Veurem com configurar un clúster de bases de dades MongoDB en el níquid i com fer consultes sobre les dades emmagatzemades.

En el següent vídeo pots veure un resum del que tractarem en aquest lliurament.

SBD L1: Què és Big Data. Bases de dades NoSQL



Vídeo: Resum del Lliurament 1

## 2. Què és Big Data?

En aquest apartat farem una breu introducció històrica a les dades que recopilam i analitzam des de temps molt antics, primer en suports físics i molt recentment de forma digital. Acabarem veient les característiques que defineixen el que entenem per *big data* o dades massives.

## 2.1. Les dades: una mica d'història

El gran historiador grec Tucídides va narrar les guerres del Peloponès. Ens aturarem en un dels episodis més famosos, el setge de Platea. En l'any 429 a.C., les tropes del rei Arquidam II d'Esparta varen assetjar la ciutat de Platea, aliada d'Atenes. Després de nombrosos intents infructuosos de trencar les defenses, els espartans varen decidir construir un mur que envoltava tota la ciutat, bloquejant-la completament durant gairebé dos anys.



imatge: Imatge idealitzada del setge de Platea. Font: alamy.com

Els defensors, cansats d'esperar l'ajuda d'Atenes, que mai no va arribar, varen concebre un pla desesperat per poder fugir-hi: construir unes escales per poder escalar el mur. Però de quina alçada havien de ser aquestes escales? Com que no podien acostar-se al mur sense arriscar les seves vides, saber-ne l'alçada no era fàcil.

Tot i que el mur estava cobert per morter, varen observar que en una zona encara es podien veure els maons. Coneixent el nombre de fileres de maons que formava el mur, podrien calcular la seva alçada. Però comptar les fileres de maons a tal distància induceix inevitablement a errors. Així que varen comanar la tasca a un gran nombre de soldats. Una vegada recopilades les dades, varen triar el resultat més repetit (el que ara denominam *moda*) i varen construir les escales d'aquesta alçada. Finalment, en una nit de tempesta varen poder fugir 212 defensors emprant les escales, i arribar fins a Atenes.

Aquest és, probablement, el primer cas de recopilació i anàlisi de dades del qual tenim constància.

En realitat, totes les societats des de la Prehistòria han recopilat dades: de població, d'extensió de finques, de caps de bestiar, de producció de cultius, etc. Són els denominats **censos**, que des d'èpoques molt llunyanes fan els governs per recollir dades, no només sobre el nombre de persones, sinó sobre diferents aspectes de l'activitat agrícola, industrial i comercial.

A partir del segle XVII podem trobar exemples més científics d'una anàlisi estadística d'aquestes dades. El comerciant anglès John Graunt va emprar les taules de mortalitat (*Bills of Mortality*), un registre setmanal de totes les defuncions i les seves causes que ocorrien a Londres, per fer un sistema de detecció precoç de brots de pesta bubònica. A més, va fer diverses investigacions sobre els factors biològics i socioeconòmics de la mortalitat i les conseqüències demogràfiques i socials que se'n deriven. En la mateixa època, l'alemany Gaspar Neumann va demostrar, basant-se en els registres de defuncions, que la creença popular que deia que en els anys acabats en 7 moria més gent, era falsa.

Aquests exemples mostren que les dades comencen a emprar-se no només per representar una realitat, sinó per extreure'n conclusions. És el naixement de l'estadística moderna, que avançarà notablement amb el desenvolupament del mètode científic durant el segle XVIII. A partir d'aleshores, van establint-se els fonaments matemàtics dels **mètodes estadístics**, amb les aportacions de grans matemàtics com Gauss, Laplace o Legendre. També són de gran importància els treballs de Galton i Pearson, que a cavall entre els segles XIX i XX, defineixen els conceptes de **correlació i regressió**, tan utilitzats avui en dia, i que suposen el pas de l'estadística deductiva a l'estadística inductiva. És a dir, no només treure conclusions del passat, sinó intentar predir el futur.

En el Lliurament 2 veurem els fonaments de l'estadística que necessitarem per poder analitzar les dades.

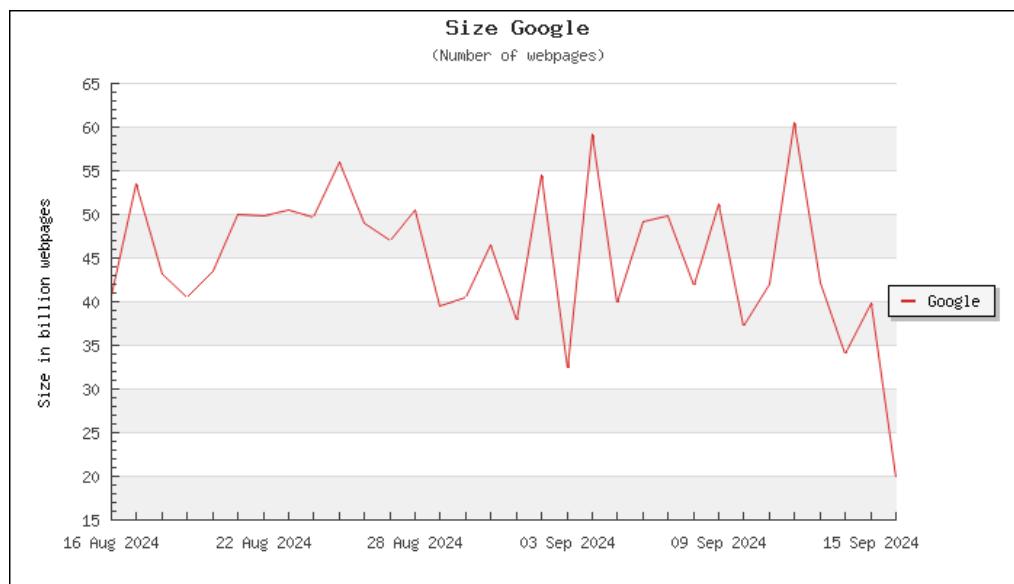
Ja hem comentat que l'elaboració dels cens nacionals ha tingut una gran importància en el desenvolupament de l'estadística. Però val la pena mencionar que també ha estat important en el desenvolupament de la informàtica. De cada vegada, el número de variables que es volen recollir de cada ciutadà és major, la qual cosa fa que el volum de dades a recollir, emmagatzemar i processar acabi essent molt gran, especialment en països molt poblats. Així, desenvolupar sistemes per gestionar totes aquestes dades ha estat una prioritat per als governs des de fa temps. Per exemple, per elaborar el cens d'Estats Units de 1880, es varen necessitar 8 anys. Tenint en compte que el cens se'n feia (i se segueix fent) cada 10 anys, això era un gran problema, quasi se solapava l'elaboració d'un cens amb el següent. Herman Hollerith va elaborar una màquina (mecànica, no digital), que fent servir unes targetes perforades per emmagatzemar i processar les dades, va aconseguir en 1890 reduir el temps d'elaboració del cens a només un any. Com a curiositat, Hollerith va vendre la seva màquina a una companyia que va acabar convertint-se en IBM.

## 2.2. Les dades en l'era digital

Abans de l'ús generalitzat dels ordinadors, les dades del cens, d'experiments científics, d'estudis de camp, etc., es registraven en paper, un procés molt costós, tant en temps com en doblers. Per fer la recollida de dades s'havia de fer un estudi previ de què i com es volia recollir. Les dades resultants seguien una estructura molt rígida i es representaven en paper mitjançant taules, sobre les quals s'aplicaven mètodes d'anàlisi estadística tradicional. L'ús d'ordinadors durant la segona meitat del segle XX va millorar molt tot aquest procés, fent-lo digital, però pel que als tipus de dades i els mètodes d'anàlisi no hi ha gaire canvis: les dades segueixen essent completament estructurades i se segueixen emprant els mateixos mètodes de l'estadística tradicional.

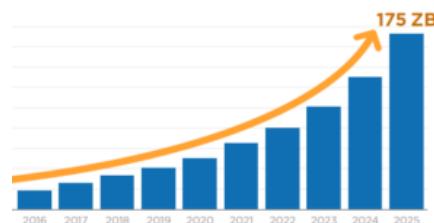
L'aparició de la web i el seu desenvolupament suposa el que s'ha anomenada l'**explosió de dades**. S'ha d'afrontar seriosament el problema de com manejar aquest immens i creixent volum de dades que es generen cada minut. Com a conseqüència apareixen nous mètodes de produir, recolectar, emmagatzemar i analitzar les dades.

Algunes estimacions ens poden ajudar a entendre la descomunal mida de la Web i Internet. Per una banda, es calcula que actualment (setembre de 2024) Google té indexades al voltant de 60.000 milions de pàgines web (<https://www.worldwidewebsize.com>)



Imatge: Nombre de pàgines indexades per Google (en milers de milions). Font: [worldwidewebsize.com](https://www.worldwidewebsize.com)

Per altra banda, segons un informe d'International Digital Corporation (IDC), el major proveïdor mundial de *business intelligence*, el volum de dades global (*datasfera*) passarà dels 64 zettabytes de 2020, als 175 zettabytes en 2025. El ritme de creixement és d'un 23% anual. Podeu trobar la taula de les unitats d'emmagatzematge més avall.



Imatge: Mida de la datasfera. Font: [datanami.com](https://datanami.com)

Es calcula que 4.950 milions de persones varen usar Internet durant el mes de gener de 2022 (62,5% de la població mundial). D'acord amb Statista, en 2025 s'enviaran uns 376 milers de milions de mails diaris. I segons

prediu IOT Analytics, en 2025 hi haurà 27,1 milers de milions de dispositius IoT (Internet of Things).

IMPORTANT

Nom	Símbol	Nombre de bytes
Kilobyte	kB	$10^3$ (1.000)
Megabyte	MB	$10^6$ (1.000 <sup>2</sup> )
Gigabyte	GB	$10^9$ (1.000 <sup>3</sup> )
Terabyte	TB	$10^{12}$ (1.000 <sup>4</sup> )
Petabyte	PB	$10^{15}$ (1.000 <sup>5</sup> )
Exabyte	EB	$10^{18}$ (1.000 <sup>6</sup> )
Zettabyte	ZB	$10^{21}$ (1.000 <sup>7</sup> )
Yottabyte	YB	$10^{25}$ (1.000 <sup>8</sup> )

**Taula:** Unitats d'emmagatzematge d'informació, segons la Comissió Electrotècnica Internacional (IEC)

Un factor molt important és que al voltant d'un 80% d'aquestes dades globals són no estructurades i apareixen en forma de text, fotos, imatges, vídeos, etc, amb les quals, els mètodes tradicionals d'anàlisi de dades no poden fer-hi res. Per això, quan xerram de *big data* o dades massives, ens referim no només a un gran volum de dades, sinó també a conjunts de dades complexes, no estructurats, de fonts variades, i per als quals fan falta noves tècniques per extreure'n informació útil.

A continuació veurem alguns exemples de fonts de dades massives.

### Pagaments amb targeta de crèdit

Un dels exemples més coneguts en l'àmbit del *big data* és el de l'anàlisi de les dades de pagaments amb targetes de crèdit. L'any 2020, només de VISA, hi havia 1.156 milions de targetes. En l'any 2018 es varen fer 368.000 milions de transaccions amb targeta al món, aproximadament uns mil milions al dia. Es dediquen grans esforços per detectar i prevenir frauds. Es fan servir tècniques de *machine learning*, tant supervisat com no supervisat, per intentar detectar situacions que se surtin del patró habitual de consum que segueix cada usuari.

### Cerques a la web i xarxes socials

Google és, amb molta diferència el motor de cerques d'Internet més utilitzat. S'estima que rep unes 99.000 cerques per segon (dades de 2024, segons demandsage.com), un 91,9% del total de cerques a Internet. De fet, la pàgina home de Google també és la pàgina més visitada de la web: uns 84.200 milions de visites mensuals (desembre 2023, segons statista.com). Com a curiositat, la paraula més cercada durant el darrer mes (setembre de 2024, segons explodingtopics.com) ha estat "YouTube", amb més de 1.200 milions de cerques.

Quan feim una cerca, s'emmagatzema una informació que pot tenir un alt interès comercial: quin terme s'ha cercat, des de quina IP, en quin moment del dia, a quines pàgines de resultat s'hi accedeix i en en quin ordre, etc. Cada clic queda registrat per al seu ús futur: els registres de seqüències de clicks (*clickstreams*) anoten la ruta seguida al llarg de la visita a distints llocs web, així com el recorregut que es realitza en l'interior de cada un d'aquests llocs. Aquestes dades es poden utilitzar per avaluar l'efectivitat de la publicitat o per detectar activitats malicioses com ara suplantacions d'identitat.

Per altra banda, les cookies també són una font important de dades massives: se solen utilitzar per registrar dades de *clickstreams* amb la finalitat d'inferir les preferències de l'usuari i enviar-li publicitat personalitzada.

Les xarxes socials són també una font immensa de dades. Pensem per exemple, en els missatges de X (antic Twitter). Segons dades de 2022, cada segon s'envien uns 6.000 tweets (posts) de mitjana, el que suposa uns 500 milions per dia, uns 200 milers de milions per any.

Un altre gegant és Facebook, que amb els seus més de 2.200 milions d'usuaris actius, genera 4 petabytes de dades per dia. Tot això s'emmagatzema en un Hive (veurem aquesta eina en el mòdul de Big Data Aplicat), que

conté al voltant de 300 petabytes.

Una de les àrees que més han crescut durant els darrers anys és l'anàlisi d'emocions en les xarxes socials: extreure automàticament mitjançant l'anàlisi de dades els sentiments que desprenen els comentaris dels usuaris en les xarxes socials, com ara ira, amor, odi, sorpresa, vergonya, eufòria, etc.

## Dades mèdiques

Els expedients mèdics ja estan, en una gran part, digitalitzats. En general, es tracta d'informació estructurada. Però a més d'aquest tipus d'informació, els dispositius portables permeten monitorar diferents paràmetres d'interès mèdic. Pensem per exemple, en les polseres d'activitat, tan populars avui dia. Permeten recollir dades de ritme cardíac, pressió sanguínea, patrons de son, nombre de passos caminats, etc. Aquesta informació pot tenir interès per a, per exemple, una companyia asseguradora. De fet, hi ha pòlies de vida que regalen una pulsera d'aquest tipus en subscriure-la. Deixant de banda les qüestions ètiques, també per a una empresa pot ser interessant tenir aquesta informació, per controlar la salut dels seus empleats. I n'hi ha que ofereixen incentius als seus empleats per utilitzar aquests dispositius i aconseguir determinats objectius.

En tot cas, aquestes dades són ínfimes si les comparen amb el major magatzem de dades que existeix avui en dia: el de la seqüenciació del genoma humà. Anualment es produeixen les dades d'uns 220 milions de genomes, que suposen uns 40 exabytes. Això és més de 40 vegades la quantitat de dades que produeix Youtube en el mateix període de temps.

## Dades científiques

Un clar exemple de dades massives en l'àmbit científic és el del Gran Colisionador d'Hadrons (LHC) del CERN. En el LHC, es produeixen 40 milions de collisions de grups de protons cada segon, la qual cosa suposa uns 1.500 bilions ( $1,5 \times 10^{15}$ ) de collisions a l'any. El volum actual de dades és d'uns 260 petabytes. Per poder gestionar aquesta informació, el CERN utilitza un grid d'aproximadament 150.000 nuclis de processament (que poden arribar a 250.000 en pics) distribuïts geogràficament entre més de 200 universitats i centres de recerca de 45 països.

Els grans telescopis també són una font d'un volum enorme de dades. Per exemple, el Very Large Telescope, format per un conjunt de 4 telescopis òptics situats a Chile, produeix uns 15 Tb de dades cada nit. Un altre exemple és el Vera C. Rubin Observatory (abans anomenat Large Synoptic Survey Telescope) un observatori astronòmic que s'està construint, també a Chile, amb l'objectiu d'elaborar mapes successius del cel nocturn durant 10 anys. S'estima que recollirà uns 30 Tb d'imatges cada nit. Per emmagatzemar aquestes imatges es necessitarà una base de dades de 60 petabytes. La taula principal de la base de dades contindrà 20.000 millions de files.

I no només telescopis òptics: el Square Kilometer Array (SKA) un radiotelescopi que s'està construint entre Austràlia i Sudàfrica i que tindrà una superfície total d'un  $\text{km}^2$ , s'estima que haurà de recollir uns 160 Tb de dades en brut per segon en els seus inicis, i encara més en fases posteriors.

## Dades en temps real

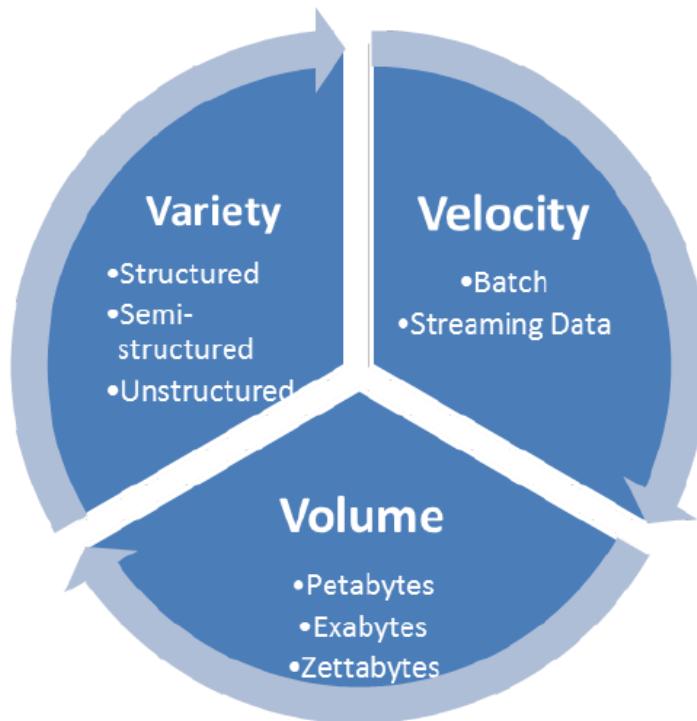
Estam envoltats de sensors. Ja hem comentat que en 2025 hi haurà més de 27 milers de milions de dispositius IoT. Un exemple són els sensors de la qualitat de l'aire que tenim a les nostres ciutats que ofereixen informació sobre diversos paràmetres en temps real.

Però un dels casos més interessants pel que fa al *big data* és el dels vehicles autònoms. Aquests vehicles han de ser capaços de recopilar dades del seu entorn, processar-les i, a partir d'elles, prendre decisions en temps real. S'estima que un vehicle autònom genera, de mitjana, uns 30 Tb de dades diàries. Un àrea important dins del *big data* és el que s'anomena d'anàlisi de dades en temps real (*stream analytics*).

## 2.3. Definició de Big Data: les tres V (o més)

La traducció de *Big Data* al català més habitual és la de *dades massives* (*datos masivos* en castellà). Però realment, què entenem per *massives*? Evidentment, el volum és un factor important, però no l'únic.

No hi ha un consens en la definició d'aquest concepte, però la més estesa és la definició de les tres V donada en 2001 per Doug Laney, de l'empresa d'anàlisi Gartner, que es basa en tres característiques o magnituds principals (totes comencen per la lletra V, tant en anglès com en català): volum, varietat i velocitat.



Imatge: Les 3 V del big data. Font: researchgate.net

### Volum

La primera característica d'aquests sistemes fa referència a la immensa quantitat de dades que es generen i recopilen constantment. I el principal repte de les solucions de *big data* és, en conseqüència, poder emmagatzemar i processar aquestes enormes quantitats de dades.

Hi ha conjunts de dades que ningú dubta que són grans. N'hem comentat uns quants d'ells en l'apartat anterior.

Però no és fàcil posar un límít a partir del qual es pugui considerar que el volum de dades és *big* o *massiu*. En 2012, un estudi internacional, en el qual varen participar més de 1.000 experts, demanava quina seria la mida per considerar un conjunt de dades com a gran. La meitat varen contestar que entre 1 terabyte i 1 petabit, mentre que un terç va respondre "no ho sé". A més, l'estudi també demanava triar una característica definitòria del que entenem per *big data*. Només un 10% va marcar l'opció "grans volums de dades".

Per altra banda, el que avui es pot considerar com a gran, en 10 anys, amb més i millors sensors que recullen més dades, i amb millors dispositius d'emmagatzematge, pot deixar de ser-ho.

Així doncs, fent una aproximació més flexible, podem considerar que el criteri del volum se compleix si el conjunt de dades és tal que no resulta viable reunir-lo, emmagatzemar-lo i analitzar-lo mitjançant els mètodes estadístics i informàtics tradicionals.

### Varietat

El *big data* de les organitzacions no està conformat només per dades internes creades per la pròpia 'organització, sinó que sovint es fan servir dades externes. La connexió a la Xarxa ens dona accés a un immens conjunt de dades procedents de fonts molt diverses. Hem de tenir present que s'estima que en l'any 2020 es varen superar els 25.000 milions de dispositius connectats a Internet, arribant a més de 60 zettabytes de dades. Ja hem comentat en l'apartat anterior el gran volum de missatges de Twitter (uns 200 milers de milions de tweets per any). I aquests missatges poden tenir un important valor comercial. Altres exemples de fonts de dades, a més dels missatges en xarxes socials, poden ser les interaccions amb els clients, fitxers de log, dades generades automàticament per sensors, localització de telèfons mòbils, documents de tot tipus, etc. I no només dades textuales, sinó també imatges, vídeos, àudio, etc.

En definitiva, xerram d'unes dades molt diverses, de diferents fonts i formats, amb redundàncies i errors. Aquesta situació és molt diferent de les dades netes i precises amb les que treballen els mètodes estadístics tradicionals. Seguint amb l'exemple de Twitter, els missatges solen analitzar-se depenent de si expressen sensacions positives, negatives o neutres. Aquesta nova àrea de l'anàlisi de sentiments requereix tècniques desenvolupades de manera específica, i només podrà ser eficaç si es treballa amb eines d'anàlisi orientades a dades massives.

En general, podem classificar les dades en tres categories:

- **Estructurades:** les dades segueixen un esquema fixat, on els valors tenen un format, longitud i significat ben definit. Normalment, s'emmagatzemem en taules de bases de dades relacionals o, si el volum és gran, en magatzems de dades (*data warehouses*)
- **No estructurades:** les dades no tenen un format ni una mida predefinida, no s'ajusten a un esquema fixat. És habitual emmagatzemar aquestes dades en bases de dades NoSQL
- **Semiestructurades:** combinen dades estructurades i no estructurades

La immensa majoria de les dades massives procedents de la Xarxa són no estructurades.

## Velocitat

Tal i com hem comentat, les dades flueixen contínuament des de fons com la Xarxa, els telèfons intel·ligents o diferents tipus de sensors. Per velocitat ens referim a la rapidesa en què aquests fluxos de dades es van generant. La velocitat és una característica necessàriament molt lligada al volum: com més ràpid es generen dades, més quantitat n'hi ha.

Per altra banda, la velocitat no només té a veure amb la rapidesa en què es generen les dades, sinó en la qual es procesen. Molt sovint, l'anàlisi de les dades es fa mitjançant processos ***batch***, que poden durar hores o, fins i tot, dies. En canvi, perquè un vehicle autònom sigui fiable, les dades procedents dels seus sensors han de processar-se, obligatòriament, en ***temps real***.

La variabilitat, entesa com els canvis de ritme en la generació del flux de dades són un aspecte a tenir en compte, en relació a la velocitat. Un exemple són els increments de tràfic de dades en hores punta. Aquest tret és important, ja que els sistemes tendeixen a fallar més durant aquests episodis. Hi ha autors que la consideren com una característica independent (una altra V).

## Altres V

A les tres V originals de Laney, altres autors han anat afegint altres característiques a la definició de dades massives. Curiosament, tots s'han obstinat en què sempre comencin per la lletra V. Alguns exemples són veracitat, visualització, valor, variabilitat (que ja hem comentat dins velocitat), vulnerabilitat, viabilitat, o fins i tot, viscositat i volatilitat.

De totes aquestes, la més referenciada és la de ***veracitat***, que fa referència a la qualitat de les dades recollides. L'anàlisi estadística tradicional fa feina amb dades precises i fiables. En canvi, les dades que es produeixen en el marc del *big data*, com ja hem comentat, sovint no tenen estructura i no s'han dissenyat pensant en l'anàlisi que posteriorment se'n pugui fer d'elles. I tot i així, volem extreure'n informació rellevant. Si prenem l'exemple dels missatges de les xarxes socials, podem veure que són dades imprecises, incertes, i amb freqüència

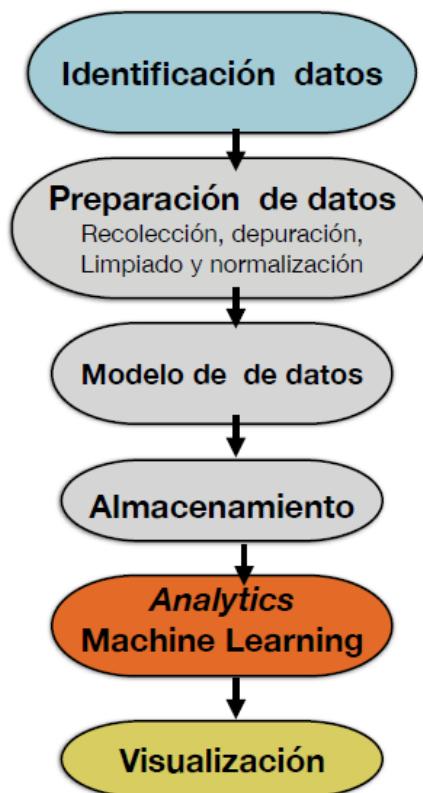
voluntàriament falses. El volum pot ajudar a detectar les dades que volem descartar. Però també el volum, de vegades, té un efecte negatiu ja que pot fer obtenir correlacions irrellevants.

Altres dues que també són rellevants són **valor** i **visualització**. El valor sol fer referència a la qualitat dels resultats que es poden extreure de l'anàlisi de les dades massives. De fet, aquest valor es pot mesurar econòmicament, ja que es venen i compren conjunts de dades. "Les dades són el nou petroli" és una frase, atribuïda a Clive Humby en 2006, que resumeix aquesta idea. Per altra banda, la visualització no és una característica intrínseca de les dades massives. Però sí que és fonamental per poder presentar-les i comunicar els resultats de l'anàlisi que se'n fa d'elles.

### 3. Sistemes de Big Data

Quan treballam amb dades massives, seguim una sèrie d'etapes amb l'objectiu d'extreure informació rellevant a partir de les dades. Són les següents:

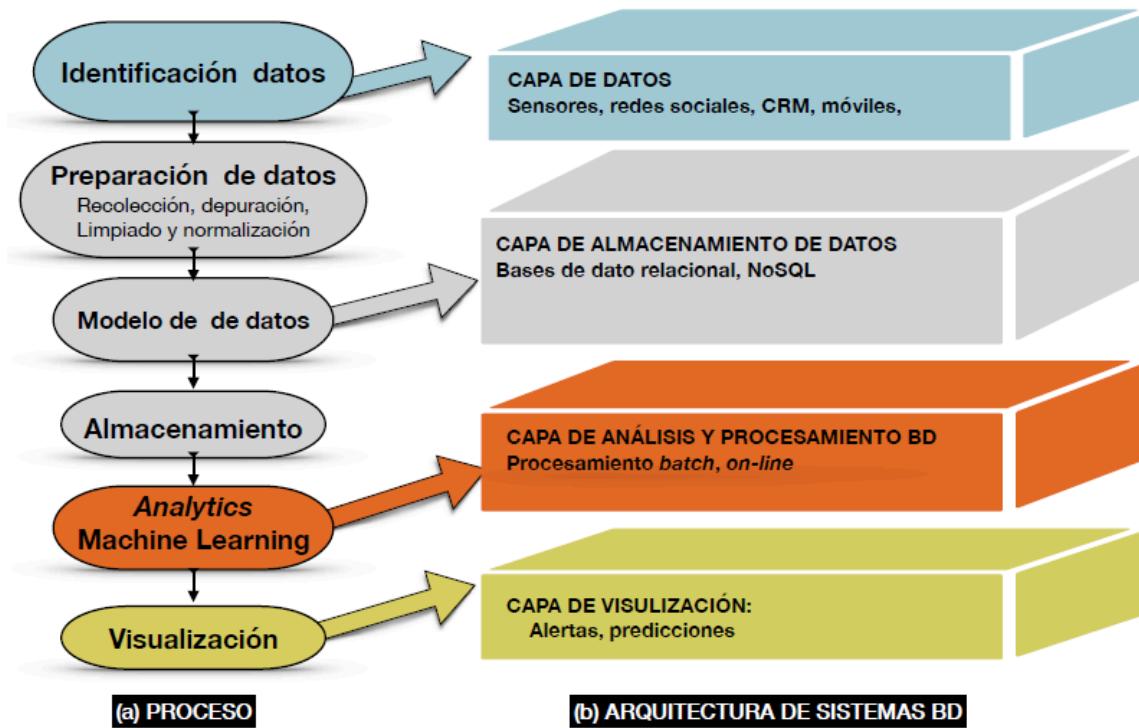
1. **Identificació de dades:** En aquesta fase s'identifiquen totes les fonts generadores de dades que estan disponibles i que són d'utilitat per a l'objectiu del projecte.
2. **Processament de les dades**, que consta de tres subfases:
  - **Preparació o pre-processament:** en un primer moment tenim les dades en *cru*, tal com ens venen de les fonts. El que volem ara és millorar la qualitat d'aquestes dades. Aplicarem diverses tècniques per detectar i eliminar dades poc fiables o defectuosos, homogeneitzar valors i detectar dades absents.
  - **Modelat:** es defineix un model de dades amb l'objectiu d'organitzar, classificar i especificar com es relacionen les dades existents. També s'estableixen les condicions que han de complir les dades per a reflectir la realitat modelada i considerar-les vàlides. També es defineixen les transformacions necessàries que s'han de realitzar per a poder ser utilitzades.
  - **Emmagatzematge:** es procedeix a la implementació del model en una base de dades o un magatzem de dades.
3. **Anàlisi:** mitjançant mètodes estadístics i tècniques d'aprenentatge automàtic se cerquen patrons d'interès, es determinen tendències, s'identifiquen anomalies, etc.
4. **Visualització:** els resultats obtinguts es lliuren, normalment de manera gràfica i interactiva, per a la seva interpretació i documentació. Aquesta informació resultant és la que servirà als humans per a la presa de decisions. Un exemple notable de visualització són els anomenats quadres de comandament integrals, que monitoren en temps real l'activitat d'una empresa o organització i permeten prendre decisions a curt termini adequades a l'estratègia marc adoptada.



**Imatge:** Procés d'extracció d'informació de les dades. Font: Universidad de Castilla-La Mancha

Un **sistema de Big Data** normalment s'estructura en diverses capes, una per a cada una de les etapes del procés que acabam de descriure. La capa de dades recol·lecta les dades provinents de fonts diverses com a xarxes socials, sensors o de la pròpia organització. La següent capa és la de l'emmagatzematge, on s'integra i s'emmagatzema en un repositori de dades (una base de dades relacional o NoSQL o un magatzem de dades).

La següent capa és la d'anàlisi, on s'extreu la informació rellevant. Finalment, en la capa de visualització es mostren els resultats a l'usuari.



**imatge:** Estructura multicapa dels sistemes de Big Data. Font: Universidad de Castilla-La Mancha

En els següents apartats d'aquest lliurament ens centrarem en la capa d'emmagatzematge. S'assumeix que les bases de dades relacionals són prou coneudes pels alumnes, així que veurem què són les bases de dades NoSQL. En aquest lliurament ens centrarem en les bases de dades basades en documents, mentre que en el Lliurament 3 veurem en detall les basades en grafs. En el Lliurament 4, descriurem què són els sistemes analítics o OLAP (processament analític en línia - *on-line analytical processing*), front als més coneguts sistemes transaccionals o OLTP (processament de transaccions en línia - *on-line transaction processing*).

Per altra banda, el comportament dels sistemes de Big Data es pot dividir en dues classes:

- **Batch Processing:** Processament per lots de dades en repòs. En aquest escenari, les dades d'origen es carreguen en els dispositius d'emmagatzematge de dades, ja sigui per la pròpia aplicació d'origen o per un flux de treball d'orquestració (*orchestration workflow*). A continuació, les dades es processen en el lloc per un treball parallelitzat, que també pot ser iniciat pel flux de treball d'orquestració.
- **Stream Processing:** Processament continu de dades en temps real, és a dir, en quant existeix disponibilitat de dades, aquestes es processen de manera seqüencial. S'estableixen uns fluxos de dades infinites i sense límits de temps.

## 4. Bases de dades NoSQL

Tot i que originalment NoSQL es refereix a qualsevol bases de dades no relacional, el terme s'ha fet servir bàsicament per referir-se als gestors que han aparegut durant els darrers anys, orientats a satisfer les necessitats de les grans companyies de la web, especialment en l'àmbit d'aplicacions de dades massives, en temps real i d'alta disponibilitat.

NoSQL no és un model de bases de dades, com pugui ser el relacional, sinó que és un concepte que aglutina diferents models (clau-valor, basat en documents, columnar, o basat en grafs). Normalment permeten una definició de dades flexible (*schemaless*, sense esquema de dades), on no totes les instàncies d'una col·lecció tenen els mateixos atributs. A més, en general, són distribuïdes, afavorint les característiques de disponibilitat i tolerància a particions, front a la consistència. Per aquesta raó, en general, no solen garantir les propietats del model ACID (atomicitat, consistència, aïllament i durabilitat). També solen ser solucions de codi obert.

### El teorema CAP

El teorema CAP, formulat per Eric Brewer, diu que un sistema de dades distribuïdes no pot garantir simultàniament les següents característiques:

- **Consistència (C - Consistency)**: qualsevol petició de lectura rep com a resposta l'escriptura més recent. És a dir, tots els clients veuen la mateixa dada al mateix temps, independentment del node al qual es connectin.
- **Disponibilitat (A - Availability)**: qualsevol petició a un node actiu rep sempre una resposta vàlida, encara que hi hagi un o diversos nodes inactius. És a dir, tots els nodes del sistema retornen una resposta vàlida per a qualsevol petició, sense cap excepció.
- **Tolerància a les particions (P - Partition Tolerance)**: el sistema ha de continuar operant encara que hi hagi una partició de la xarxa (una interrupció de la comunicació entre els nodes de la xarxa, que pot provocar que es perdin o retardin un número arbitrari de missatges).

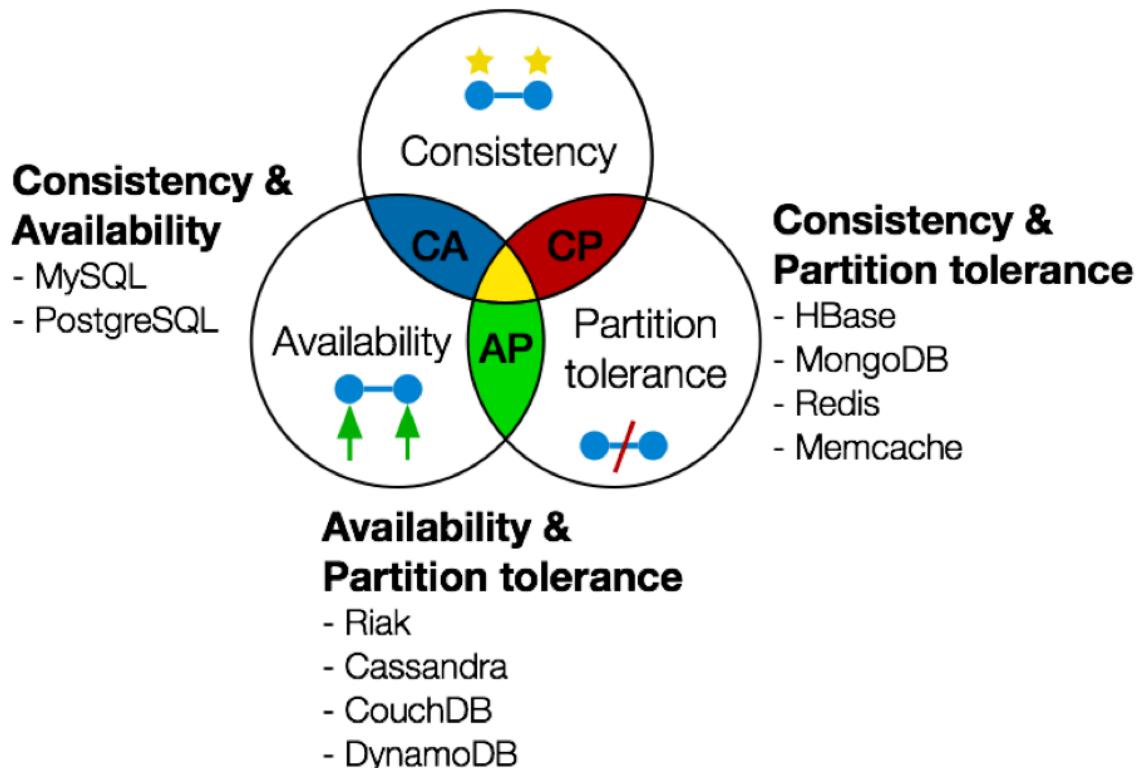
Com que cap sistema pot cumplir les tres característiques, s'ha de centrar en satisfer almenys dues d'elles.

Les bases de dades relacionals se centren en les propietats de consistència (C) i disponibilitat (A), però no suporten les particions. És el que anomenam un **enfocament CA**.

En canvi, un dels requisits dels les bases de dades NoSQL és que siguin tolerants a les particions. Per tant, en podem trobar dos enfocaments diferents, depenent de quina sigui l'altra propietat que satisfacin:

- **Enfocament AP** (disponibilitat i tolerància a les particions): el sistema sempre està disponible, tot i que temporalment pot mostrar dades inconsistents en presència de particions.
- **Enfocament CP** (consistència i tolerància a les particions): el sistema sempre conté una visió consistent de les dades, tot i que pot no estar totalment disponible en presència de particions.

La següent imatge mostra aquests conceptes que hem introduït i alguns exemples representatius de sistemes gestors de bases de dades de cada un dels tres enfocaments:



Imatge: Teorema CAP. Font: asesoftware.com

A continuació veurem els fonaments dels quatre models diferents que hem enumerat abans:

- clau-valor
- basat en documents
- columnar o *wide-column*
- basat en grafs

És important mencionar que alguns dels gestors de bases de dades NoSQL són **multi-model**, és a dir, suporten més d'un d'aquests models.

Acabarem el tema veient amb més profunditat el gestor de bases de dades NoSQL més utilitzat, MongoDB, que segueix un model basat en documents.

## 4.1. Model clau-valor

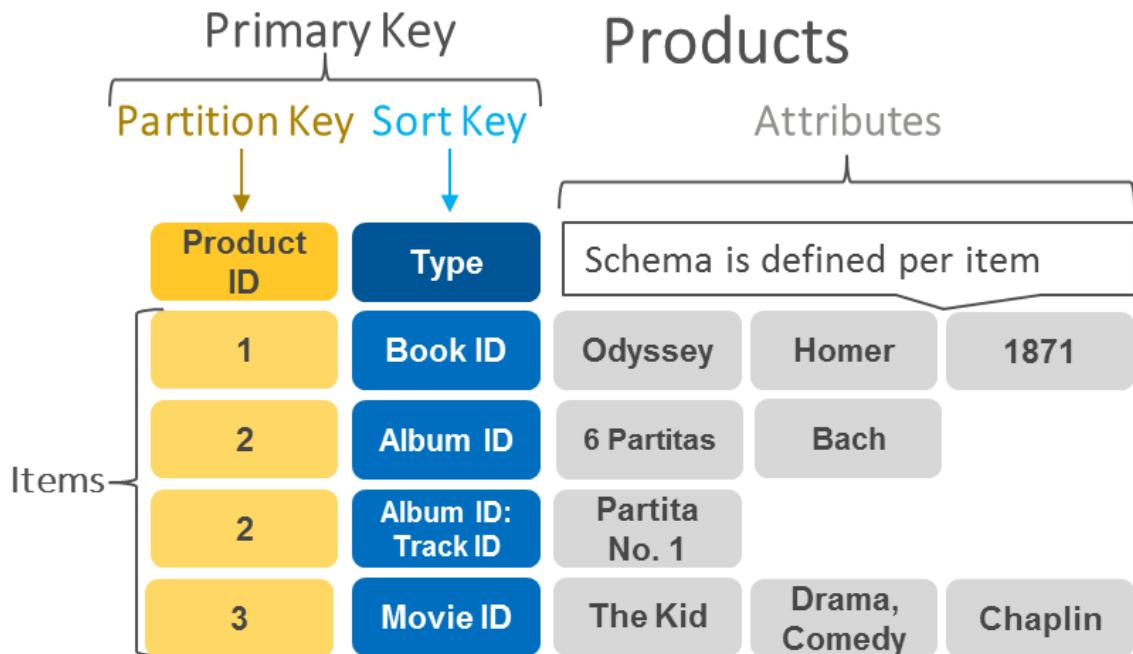
En una base de dades clau-valor, definim una col·lecció com un conjunt d'instàncies on, de cada una, tenim una **clau**, que serveix per a identificar la instància, i un **valor**, que pot ser qualsevol informació. No definim com és aquesta informació, serà l'aplicació (i no el sistema gestor) la que s'encarregarà de tractar aquests valors de la manera convenient.

En la següent imatge, podem veure una col·lecció on cada valor té un conjunt d'atributs diferents:

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

**Imatge:** Exemple de col·lecció d'instàncies en un model clau-valor. Font: Wikipedia Commons

A continuació podem veure un altre exemple, on la clau està formada per dos atributs, un ID i un tipus. que pot ser un llibre, un àlbum (disc), una cançó o una pel·lícula:



**Imatge:** Exemple de col·lecció d'instàncies en un model clau-valor, amb una clau múltiple (amb Amazon Dynamo DB).  
Font: aws.amazon.com

Aquestes bases de dades són molt flexibles i són molt fàcilment particionables. Això les fa molt escalables, de manera que poden processar grans quantitats de dades en temps real en entorns amb milions d'usuaris.

Amazon Dynamo DB, Oracle NoSQL Database, Apache Ignite o Redis són alguns exemples de gestors que suporten aquest model.



## 4.2. Model basat en documents

En aquest model, cada una de les instàncies que s'emmagatzemem és un document, normalment JSON, tot i que també pot ser XML (o altres formats). La base de dades estarà formada per col·leccions de documents.

En la següent imatge podem veure un exemple d'un document JSON que conté les dades d'un client. Podem veure que conté diversos camps: un identificador únic de la instància, el seu nom i llinatge, la seva adreça (que és un sub-document format pels camps carrer, ciutat, estat i codi postal), i una llista de les seves aficions. En la nostra base de dades tindrem una col·lecció de documents com aquest, un per a cada client.



```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  },
  "hobbies": ["surfing", "coding"]
}
```

**imatge:** Exemple de document JSON. Font: [mongodb.com](https://www.mongodb.com)

Si ho compararem amb una base de dades relacional, una col·lecció seria l'equivalent a una taula, un document seria l'equivalent a una fila de la taula, mentre que un camp del document ho seria a una columna d'una fila. Però hi ha dues diferències fonamentals, que constitueixen els dos principis del model basat en documents, i que descrivim a continuació.

### Primer principi: Les dades que s'accedeixen conjuntament, han d'emmagatzemar-se conjuntament (en el mateix document)

Si ens fixam en l'exemple anterior, podem veure que dins les dades de la persona també apareixen les dades de la seva adreça, mitjançant un sub-document. En una base de dades relacionals, tindriem dues taules, que estarien relacionades mitjançant una clau forània. En canvi, aquí s'agrupen les dades que normalment apareixen juntes en un únic document, evitant complexos joins que poden arribar a ser ineficients.

Pensem ara un exemple una mica més complex com pot ser el carretó de la compra d'un client d'una web de comerç electrònic. En un model relacional això involucra nombroses taules (client, adreça d'enviament, adreça de facturació, producte, mitjà de pagament, etc.). En un model basat en documents, tota la informació d'un carretó es guardaria junta, com una única instància de la col·lecció, és a dir, un únic document JSON (o XML) amb tota la informació al respecte. Això fa que pugui arribar a ser molt més eficient, permetent solucions més escalables i orientades a alta disponibilitat.

S'ha de tenir en compte que el model relacional es va concebre en un moment en què s'havia de primar l'espai d'emmagatzematge, evitant la duplicació de dades. El resultat, però, sovint són esquemes formats per una enorme quantitat de taules, que són difícils d'entendre i que suposen un gran esforç per mantenir. En canvi, avui en dia el cost d'emmagatzematge ja no és una prioritat i les bases de dades NoSQL en general, i les basades en documents en particular, volen primar l'eficiència, el caràcter distribuït de les dades, la flexibilitat i la facilitat per al manteniment.

## Segon principi: Flexibilitat, adaptació al canvi

No tots els documents d'una col·lecció tenen perquè tenir la mateixa estructura. Així doncs, podríem trobar que el document JSON d'un altre client de la col·lecció té altres camps diferents. Per això es diu que són **schemaless** (o sense esquema), és a dir, que els documents no han de seguir forçosament un esquema. Si es vol, es pot obligar a què tots els documents siguin compatibles amb un esquema predeterminat, però no és el més habitual.

Això dona una gran flexibilitat als desenvolupadors, ja que, davant un canvi, no fa falta executar sentències de tipus *alter table*, simplement, els canvis queden recollits dins dels mateixos documents. Es permeten així canvis fins i tot en temps d'execució.

Una vegada que tenim les dades representades mitjançant col·leccions de documents, els diferents sistemes gestors basats en documents ofereixen la seva pròpia API per poder implementar operacions CRUD (*create, read, update, delete*). Són l'equivalent a SQL en una base de dades relacional. Un altre avantatge del model basat en documents, respecte a les bases de dades relacionals, és que aquestes darreres necessiten una capa d'ORM (*Object-Relational Mapping*), on es fa el mapeig dels objectes de la capa de negoci amb les taules de la capa de dades. És el que sovint s'anomena "desajust d'impedància" (**impedance mismatch**). Amb el model basat en documents, la traducció d'objectes a documents i viceversa és directa. Això suposa que ja no és necessari un llenguatge com SQL, que no s'ajusta al model orientat a objectes de les aplicacions, sinó que les aplicacions treballarem sempre amb objectes (i documents).

El gestor basat en documents més àmpliament utilitzat és **MongoDB**, sobre el qual aprofundirem en l'apartat 5. Altres molt coneguts són Amazon DocumentDB o Apache CouchDB.

### 4.3. Model columnar o wide-column

En aquest model també es fan servir taules, files i columnes, però a diferència de les bases de dades relacionals, cada fila pot tenir diferents columnes (different número, noms i tipus). Així doncs, cada fila conté diverses columnes (no sempre les mateixes), i per a cada una, necessitarem guardar el seu nom i valor.

En la següent imatge podem veure un exemple d'una taula amb el model columnar. Conté tres files. La primera té tres columnes A, B i C, cada una amb el seu valor; la segona fila té les columnes B, D i E; mentre que la tercera fila té dues columnes, A i F.

Fila 1	Columna A	Columna B	Columna C
	valor	valor	valor
Fila 2	Columna B	Columna D	Columna E
	valor	valor	valor
Fila 3	Columna A	Columna F	
	valor	valor	

**Imatge:** Taula amb un model columnar

Al igual que les bases de dades basades en documents, també estan orientades a agrupar dades que solen anar juntes en una única fila (com en l'exemple del carretó que hem vist abans). I també segueixen un enfocament *schemaless*.

Les més conegudes són BigTable de Google i Apache Cassandra, a més de **HBase**, que forma part de l'ecosistema Hadoop.

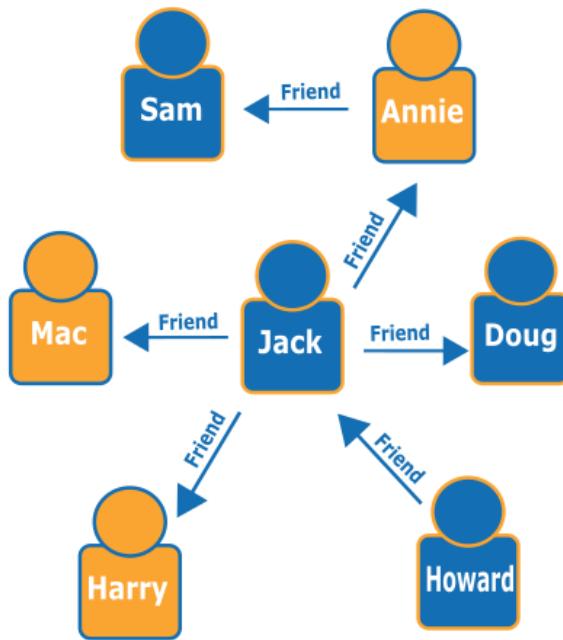
D'altra banda, el model columnar també es pot representar mitjançant el model clau-valor o el basat en documents.

Dedicarem tot el Lliurament 4 d'aquest mòdul a veure amb molt més detall les bases de dades basades columnars i, més en concret HBase.

#### 4.4. Model basat en grafs

En aquest cas, el model utilitza estructures basades en grafs, formades per nodes i arestes, per representar les dades. Està dissenyat expressament per emmagatzemar relacions i navegar per elles. És una aproximació molt diferent als models anteriors.

La imatge següent mostra el graf d'una xarxa social, amb persones (nodes) i les seves relacions d'amistat (arestes). Amb aquesta representació podem arribar a respondre preguntes com qui són els amics dels amics de Jack.



**Imatge:** Exemple de graf. Font: aws.amazon.com

Amazon Neptune, Apache Giraph, ArangoDB o Neo4J són alguns dels gestors basats en grafs més coneguts.

Dedicarem tot el Lliurament 3 d'aquest mòdul a veure amb molt més detall les bases de dades basades en grafs.

## 5. MongoDB

Tal i com hem comentat a l'apartat anterior, MongoDB és el gestor de bases de dades basades en documents més emprat. En la següent taula, extreta de [db-engines.com](https://db-engines.com), es pot veure el rànquing de popularitat dels SGBD en setembre de 2024. MongoDB ocupa la cinquena posició total, i és, amb molta diferència, el gestor NoSQL més popular.

Rank			DBMS	Database Model	Score		
Sep 2024	Aug 2024	Sep 2023			Sep 2024	Aug 2024	Sep 2023
1.	1.	1.	Oracle	Relational, Multi-model	1286.59	+28.11	+45.72
2.	2.	2.	MySQL	Relational, Multi-model	1029.49	+2.63	-82.00
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	807.76	-7.41	-94.45
4.	4.	4.	PostgreSQL	Relational, Multi-model	644.36	+6.97	+23.61
5.	5.	5.	MongoDB	Document, Multi-model	410.24	-10.74	-29.18
6.	6.	6.	Redis	Key-value, Multi-model	149.43	-3.28	-14.26
7.	7.	↑ 11.	Snowflake	Relational	133.72	-2.25	+12.83
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	128.79	-1.04	-10.20
9.	9.	↓ 8.	IBM Db2	Relational, Multi-model	123.05	+0.04	-13.67
10.	10.	↓ 9.	SQLite	Relational	103.35	-1.44	-25.85
11.	11.	↑ 12.	Apache Cassandra	Wide column, Multi-model	98.94	+1.94	-11.11
12.	12.	↓ 10.	Microsoft Access	Relational	93.76	-2.61	-34.81
13.	13.	↑ 14.	Splunk	Search engine	93.02	-3.08	+1.63
14.	↑ 15.	↑ 17.	Databricks	Multi-model	84.24	-0.22	+9.06
15.	↓ 14.	↓ 13.	MariaDB	Relational, Multi-model	83.44	-3.09	-17.01
16.	16.	↓ 15.	Microsoft Azure SQL Database	Relational, Multi-model	72.95	-2.08	-9.78
17.	17.	↓ 16.	Amazon DynamoDB	Multi-model	70.06	+1.15	-10.85
18.	↑ 19.	18.	Apache Hive	Relational	53.07	-2.17	-18.76
19.	↓ 18.	↑ 20.	Google BigQuery	Relational	52.67	-2.86	-3.80
20.	20.	↑ 21.	FileMaker	Relational	45.20	-1.47	-8.40

Imatge: Rànquing dels SGBD més populars. Font: db-engines.com

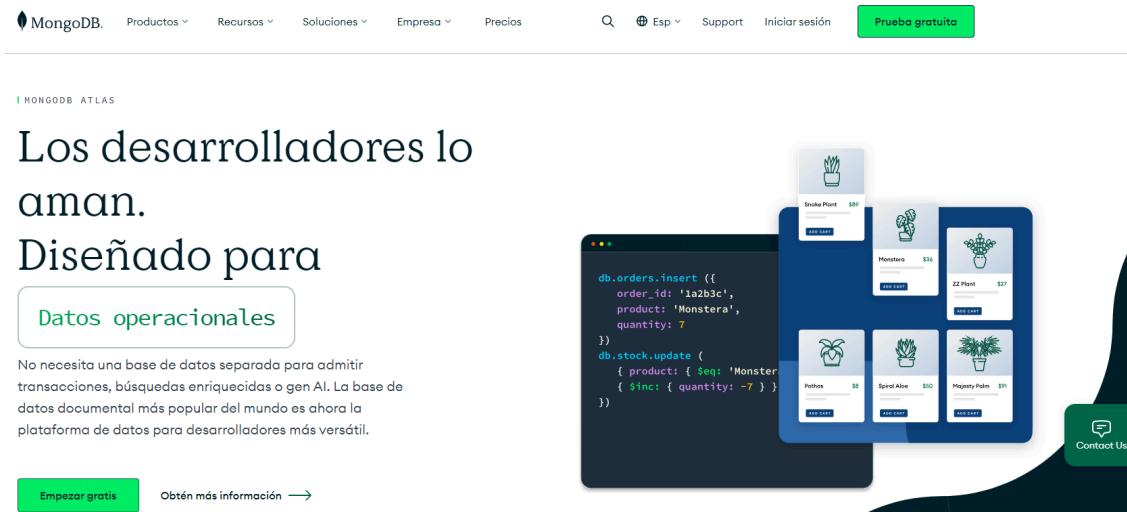
Hi ha tres factors que han ajudat a aquesta gran popularitat de MongoDB:

- el model de dades basat en documents és **intuïtiu i flexible**
- té una **arquitectura distribuïda nativa**, la qual cosa permet alta disponibilitat i escalabilitat horitzontal
- es pot executar a qualsevol lloc (*run-anywhere*): es poden crear bases de dades a les diferents **plataformes del nigul** (AWS, Azure i Google Cloud)

Una vegada que en l'apartat 4.2 hem vist els fonaments teòrics de les bases de dades basades en documents, anam ara, d'una manera pràctica, a veure el funcionament de MongoDB. En primer lloc, crearem un clúster en el nigul, mitjançant MongoDB **Atlas**, la solució de MongoDB per a *database as a service*, és a dir, per a treballar amb bases de dades al nigul. A continuació, veurem com fer algunes operacions bàsiques per interactuar amb les dades del nostre clúster, fent servir el client **Compass**, i acabarem estudiant com s'implementen les diverses operacions CRUD (*create, read, update, delete*).

## 5.1. Creació del clúster

En primer lloc entram en la pàgina de [MongoDB](#), on ens hem de registrar, de manera gratuïta. Per fer-ho, hem de pitjar al botó "Prueba gratuita", a la cantonada superior dreta. Si ho preferiu, per fer el registre podeu emprar el vostre usuari de Google. Haureu d'acceptar la política de privacitat i els termes del servei.

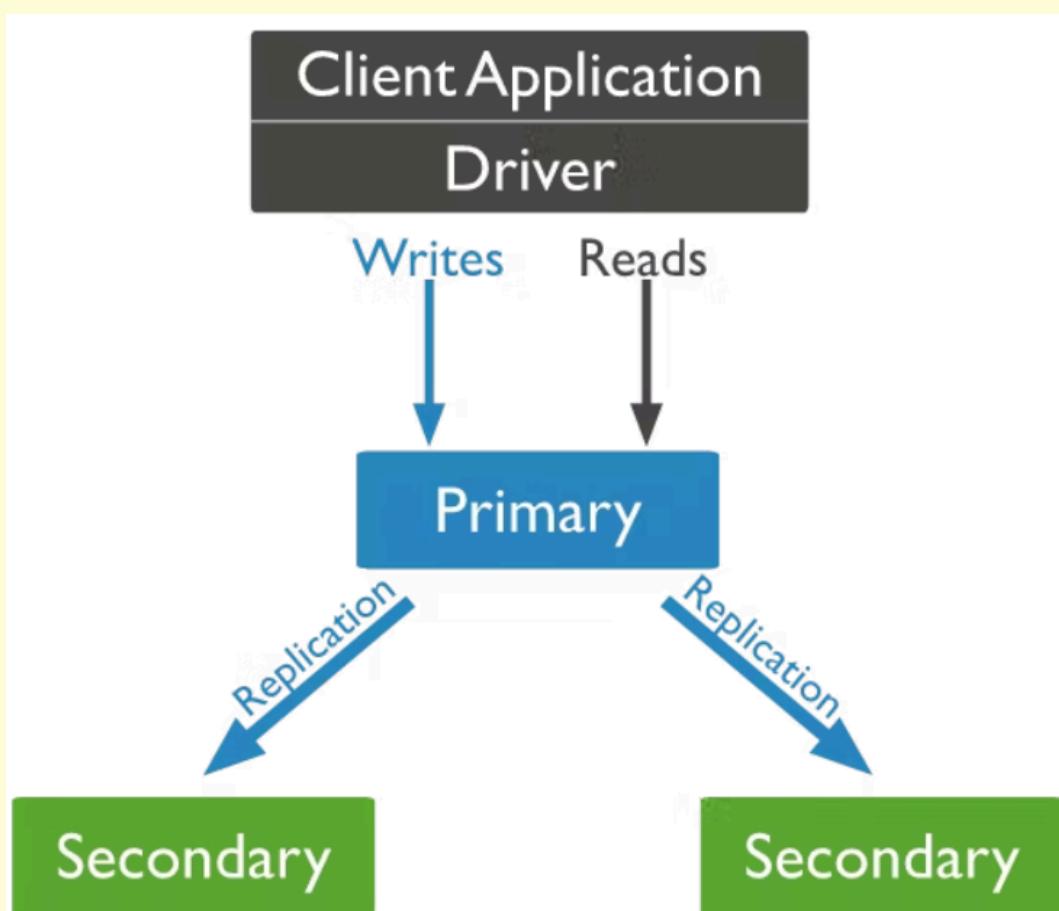


Imatge: Pàgina de MongoDB

**Un clúster** és un conjunt de servidors que treballen com si es tractés d'una única màquina.

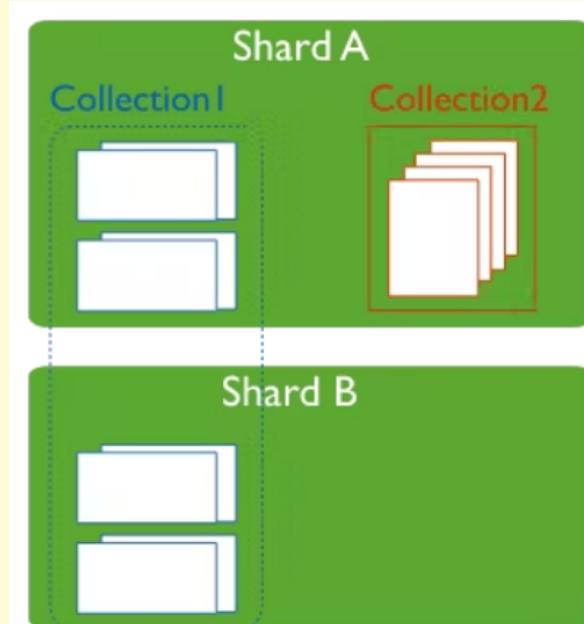
En MongoDB hi ha dos tipus de clústers: *replica sets* (conjunts de rèpliques) i *sharded cluster* (cluster fragmentat).

Un **replica set** és la replicació d'un grup de servidors MongoDB que contenen còpies de les mateixes dades. Aquesta és una propietat fonamental per als desplegaments de producció, ja que garanteix una alta disponibilitat i redundància, que són característiques crucials a tenir en compte en cas de fallides i períodes de manteniment planificats. El clúster tindrà un node primari i diversos nodes secundaris, que contenen còpies exactes del primari. Si el primari cau, es fa un procés d'elecció de quin node secundari passa a ser el nou node primari.



**Imatge:** MongoDB replica sets. Font: [mongodb.com](http://mongodb.com)

Per altra banda, un **sharded cluster** és una col·lecció de datasets distribuïts en molts servidors (*shards*), amb l'objectiu d'aconseguir una escalabilitat horitzontal i millors rendiments en les operacions de lectura i escriptura. En la següent imatge podem veure que la col·lecció 1 està fragmentada (una part està emmagatzemada al shard A i una altra al shard B), mentre que la 2 no ho està. D'aquesta manera, les lectures de dades de la col·lecció 1 es poden fer en paral·lel sobre els dos servidors: uns fragments (*chunks*) es llegiran des del servidor A i altres des del B, cosa que ens donarà un augment del rendiment (aproximadament el doble en aquest exemple).



**Imatge:** MongoDB sharded cluster. Font: [mongodb.com](http://mongodb.com)

Mentre que l'esquema de *replica sets* ens ofereix alta disponibilitat (gràcies a la redundància de dades), el *sharding* ens dona un major rendiment (mitjançant l'escalabilitat horitzontal). Aquestes dues tècniques es poden combinar, de manera que cada *shard* es configuri com a un *replica set*: un *shard* no seria un únic servidor sinó un conjunt d'ells, tots amb una rèplica de les mateixes dades (els mateixos *chunks*). Així aconseguiríem una arquitectura d'alta disponibilitat i alt rendiment.

### Què és Atlas i els clústers d'Atlas?

**Atlas** es la solució de MongoDB per oferir una *NoSQL Database-as-a-Service* al nivell públic (disponible a Microsoft Azure, Google Cloud Platform o Amazon Web Services). Atlas és un servei que permet crear i gestionar un clúster de servidors en una d'aquestes plataformes, d'una manera gràfica i senzilla, a través d'una interfície web. Atlas ens ofereix tres tipus de clústers en el nivell. Per a sistemes en producció, hauríem de triar una de les opcions de pagament. En canvi, per a aprenentatge i proves, Atlas ofereix l'opció (*M0*) de crear un clúster en un entorn compartit (*shared cluster*), de forma gratuïta. Aquests clústers són del tipus *Replica set*, amb un factor de replicació de 3 (les dades estan per triplicat).

Podeu trobar més informació sobre els clústers de MongoDB a <https://www.mongodb.com/basics/clusters>

Ens demanarà una sèrie de preguntes. Podeu triar les opcions que més s'ajustin al vostre cas. El cas més habitual serà respondre *Learn MongoDB* com al nostre objectiu principal, *I've never developed software with MongoDB before* com el temps desenvolupant amb MongoDB, *Not sure / None* en les darreres dues preguntes i el llenguatge que més us agradi.

A continuació ens demanarà després quina opció de desplegament en el nivell volem per al nostre clúster. En el nostre cas, triarem "*M0*", que és gratuïta i que ens permetrà explorar diversos datasets d'exemple. Lògicament aquesta opció només és per aprendre, no per a sistemes en producció. Després, en la mateixa pantalla, podem triar algunes opcions més específiques per al clúster. Podem deixar AWS com a proveïdor i seleccionar una regió d'Europa, millor *París (eu-west-3)*, o almenys alguna amb l'estrella que indica regió recomanada. També podem, si volem, canviar el nom del clúster, que per defecte és *Cluster0*. Finalment, pitjarem "Create Deployment" per crear el nostre clúster.

## Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

**M10**      **\$0.09/hour**

For production applications with sophisticated workload requirements.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

**Serverless**

For application development and testing, or workloads with variable traffic.

STORAGE	RAM	vCPU
Up to 1TB	Auto-scale	Auto-scale

**M0**      **Free**

For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

**Free forever!** Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

**Name**  
You cannot change the name once the cluster is created.

Automate security setup (i)  
 Preload sample dataset (i)

**Provider**  
 AWS  Google Cloud  Azure

**Region**  
 Paris (eu-west-3) ★ 🌍 (i)  
★ Recommended (i)   Low carbon emissions (i)

[I'll do this later](#)

[Go to Advanced Configuration](#)

Create Deployment

**Imatge:** Creació del clúster

Ara hem de configurar l'**usuari i contrasenya** que emprarem per accedir al clúster. Per defecte ens crea un usuari i contrasenya. Podem emprar aquest, canviar la seva contrasenya, o crear-ne un de nou. Una vegada escrits l'usuari i contrasenya, hem de fer clic a "Create Database User". En qualsevol cas, no perdeu l'usuari i contasenya que hagiu establert!

## Connect to Cluster0

1

Set up connection security

2

Choose a connection method

3

Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

### 1. Add a connection IP address

Your current IP address (80.58.145.158) has been added to enable local connectivity. Add another later in [Network Access](#).

### 2. Create a database user

This first user will have [atlasAdmin](#) permissions for this project.

We autogenerated a username and password. You can use this or create your own.

**i** You'll need your database user's credentials in the next step. Copy the database user password.

Username	Password
toni	*****
<a href="#" style="border: 1px solid #ccc; padding: 2px 10px; border-radius: 5px; color: inherit; text-decoration: none;">Create Database User</a>	
<a href="#" style="border: 1px solid #ccc; padding: 2px 10px; border-radius: 5px; color: inherit; text-decoration: none;">Close</a> <span style="margin-left: 20px;"><a href="#" style="border: 1px solid #0072BC; padding: 2px 10px; border-radius: 5px; color: inherit; text-decoration: none; background-color: #0072BC; color: white;">Copy</a></span>	

Choose a connection method

**imatge:** Usuari i contrasenya de connexió al clúster

Després de fer clic a "Choose a connection method" i a la pantalla següent podem pitjar "Close", sense seleccionar cap opció.

Amb això ja hem creat el nostre clúster (en el meu cas amb el nom "Cluster0"), dins un projecte anomenat "Project 0".

The screenshot shows the MongoDB Atlas interface for the 'Project 0' in 'TONI'S ORG'. The main area displays the 'Overview' of 'Cluster0', which has a data size of 134.44 MB. It includes sections for 'Clusters' (with 'Cluster0' selected), 'Browse collections', 'Migrate data', 'View monitoring', and 'Application Development' (set to Python). On the left sidebar, there are sections for Deployment, Services (Atlas Search, Vector Search, Stream Processing, Triggers, Migration, Data Federation, Device & Edge Sync), Security (Quickstart, Backup, Database Access, Network Access, Advanced), and New On Atlas (Goto). On the right sidebar, there are 'Featured Resources' (Python, Learn, Sample Apps) and 'New On Atlas' (Learn about the latest feature enhancements on Atlas).

**imatge:** Clúster creat

També és important **habilitar les IP** des de les quals ens connectarem. Automàticament s'ha afegit la IP des de la qual ens hem connectat. Però com que en la majoria de casos no tindrem una IP fixa, i tanmateix no tindrem dades sensibles, és convenient anar a "Nework Access", pitjar el botó "Edit" i seleccionar l'opció "Allow access from anywhere". D'aquesta manera, ens habilitarà l'adreça **0.0.0.0/0**.

The screenshot shows the MongoDB Atlas Network Access IP Access List Entry dialog box. It displays a single entry: '60.56.145.158/32' (includes your current IP address). A comment field contains 'Created as part of the Auto Setup process'. At the bottom are 'Cancel' and 'Confirm' buttons.

Imatge: Canvi de les adreces IP habilitades

Si entram en l'opció "Database" del panell de l'esquerra i anem a la pestanya "Collections", podem veure que en el procés de creació del clúster ens ha creat una base de dades de mostra anomenada "sample\_mflix", que conté 6 col·leccions de dades relacionades amb pel·lícules. Abans de continuar, volem carregar totes les dades d'exemple que ens ofereix MongoDB. Però per evitar conflictes amb aquesta base de dades "sample\_mflix" que ja tenim, anem primer a esborrar-la, pitjant el botó de la paperera que ens apareix al costat del seu nom.

The screenshot shows the MongoDB Atlas Cluster Overview page. Under the 'sample\_mflix' database, the 'comments' collection is highlighted with a red circle. Next to the collection name is a 'Drop' button, which is also circled in red. The page also shows other collections: movies, sessions, theaters, and users. The interface includes tabs for Overview, Real Time, Metrics, Collections, and others.

Imatge: Esborrat de la base de dades "sample\_mflix"

En la finestra que apareix, hem de posar el nom "sample\_mflix" i pitjar el botó "Drop". Ara ja veurem que la pestanya "Collections" apareix sense cap col·lecció i podem carregar totes les dades d'exemple fent clic al botó "Load a Sample Dataset".

**Explore Your Data**

- **Find:** run queries and interact with documents
- **Indexes:** build and manage indexes
- **Aggregation:** test aggregation pipelines
- **Search:** build search indexes

[Load a Sample Dataset](#) [Add My Own Data](#)

**imatge:** Càrrega de les dades d'exemple

En la finestra que ens apareix, hem de tornar a pitjar el botó "Load Sample Dataset". El procés de càrrega pot tardar uns minuts. Finalment, podrem veure que ens ha creat 9 bases de dades, el nom de totes elles començant per "sample". Podem veure que, entre elles, hi ha la de "sample\_mflix".

DATABASE	COLLECTIONS
sample_airbnb	9
sample_analytics	23
sample_geospatial	
sample_guides	
sample_mflix	
sample_restaurants	
sample_supplies	
sample_training	
sample_weatherdata	

**imatge:** Dades d'exemple carregades



## 5.2. Primeres passes

Dins cada una de les bases de dades d'exemple que hem carregat tenim una o més col·leccions. Per exemple, tenim una base de dades "sample\_restaurants" que té dues col·leccions, una d'elles, "restaurants" amb la informació dels restaurants de la ciutat de Nova York. Si hi feim clic podem veure les seves dades.

The screenshot shows the MongoDB Atlas interface. On the left, the sidebar is open with the 'Database' section selected. In the main area, under 'TONI'S ORG - 2024-09-16 > PROJECT 0 > DATABASES', the 'Cluster0' database is selected. Under 'Collections', the 'sample\_restaurants.restaurants' collection is highlighted. The collection details show 'STORAGE SIZE: 3.64MB', 'LOGICAL DATA SIZE: 10.30MB', 'TOTAL DOCUMENTS: 25359', and 'INDEXES TOTAL SIZE: 720KB'. Below this, there are tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. A search bar says 'Generate queries from natural language in Compass'. A query builder allows typing a query like '{ field: 'value' }'. The results section shows 'QUERY RESULTS: 1-20 OF MANY' with two documents displayed as JSON:

```

{
  "_id": ObjectId("5eb3d668b31de5d588f4292a"),
  "address": {
    "object": "Brooklyn"
  },
  "borough": "Brooklyn",
  "cuisine": "American",
  "grades": [
    "A",
    "B",
    "C",
    "D"
  ],
  "name": "Riviera Caterer",
  "restaurant_id": "40356018"
}

{
  "_id": ObjectId("5eb3d668b31de5d588f4292b"),
  "address": {
    "object": "Delicatessen"
  },
  "borough": "Brooklyn",
  "cuisine": "Delicatessen",
  "grades": [
    "A",
    "B",
    "C",
    "D",
    "E",
    "F"
  ],
  "name": "Wilken's Fine Food",
  "restaurant_id": "40356483"
}

```

**imatge:** Col·lecció restaurants

Podem desplegar els camps *address* i *grades*, per veure que el primer és un objecte (un sub-document) i l'altre un array d'objectes. Aquest és el JSON corresponent al primer restaurant de la col·lecció:

```
{
    "_id": {"$oid": "5eb3d668b31de5d588f4292a"},  

    "address": {  

        "building": "2780",  

        "coord": [  

            {"$numberDouble": "-73.98241999999999"},  

            {"$numberDouble": "40.579505"}  

        ],  

        "street": "Stillwell Avenue",  

        "zipcode": "11224"  

    },  

    "borough": "Brooklyn",  

    "cuisine": "American",  

    "grades": [  

        {  

            "date": {"$date": {"$numberLong": "1402358400000"}},  

            "grade": "A",  

            "score": {"$numberInt": "5"}  

        },  

        {  

            "date": {"$date": {"$numberLong": "1370390400000"}},  

            "grade": "A",  

            "score": {"$numberInt": "7"}  

        },  

        {  

            "date": {"$date": {"$numberLong": "1334275200000"}},  

            "grade": "A",  

            "score": {"$numberInt": "12"}  

        },  

        {  

            "date": {"$date": {"$numberLong": "1318377600000"}},  

            "grade": "A",  

            "score": {"$numberInt": "12"}  

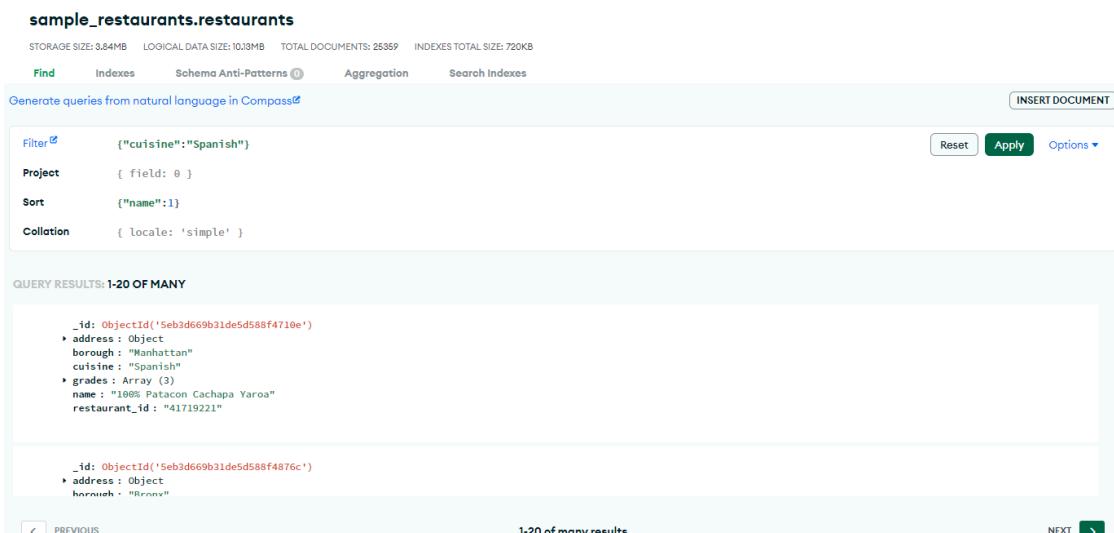
        }  

    ],  

    "name": "Riviera Caterer",  

    "restaurant_id": "40356018"
}
```

També podem escriure un filtre, de manera que només ens mostri un subconjunt de les dades. Per exemple, si escrivim `{"cuisine": "Spanish"}` en el camp "Filter" i després pitjam el botó "Apply", veurem només els restaurants de cuina espanyola. A més, pitjant a "More Options" podrem, entre d'altres, definir l'ordenació de les dades.



The screenshot shows the Compass interface for MongoDB. At the top, it displays the database and collection: `sample_restaurants.restaurants`. Below this, there are tabs for `Find`, `Indexes`, `Schema Anti-Patterns`, `Aggregation`, and `Search Indexes`. A button for `INSERT DOCUMENT` is also present. The main area contains a search bar with the query `{"cuisine": "Spanish"}`. Below the search bar, there are sections for `Project` (with a field `name`), `Sort` (with a value `1`), and `Collation` (set to 'simple'). At the bottom, there are buttons for `Reset`, `Apply`, and `Options`. The results section at the bottom shows the first 20 results of many, with the first result being a Spanish restaurant.

```

QUERY RESULTS: 1-20 OF MANY

_id: ObjectId('5eb3d669b31de5d588f4710e')
address: Object
borough: "Manhattan"
cuisine: "Spanish"
grades: Array (3)
name: "100% Patacon Cachapa Yaroa"
restaurant_id: "41719221"

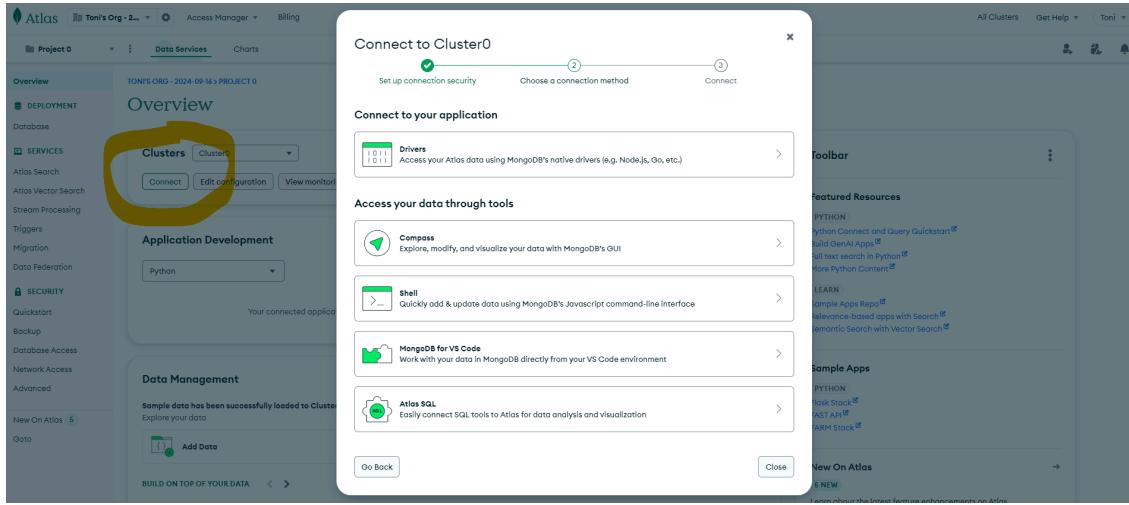
_id: ObjectId('5eb3d669b31de5d588f4876c')
address: Object
borough: "Bronx"

```

**Imatge:** Restaurants de cuina espanyola ordenats alfabèticament

Per entendre millor com podem fer operacions més complexes sobre les nostres col·leccions de dades, anem a instal·lar el programa **Compass**, que ens proporciona una interfície gràfica més completa que la interfície web d'Atlas. El podem descarregar des de <https://www.mongodb.com/products/compass>. Tria la versió estable més recent (1.44.3 en setembre de 2024) per al vostre sistema operatiu i instal·lau-la. En el cas de Windows, baixareu un fitxer .exe, que en executar-lo, us afegirà l'aplicació MongoDB Compass.

Una vegada instal·lat, l'executam i el primer que haurem de fer és afegir una nova connexió al vostre clúster. Per saber la cadena de connexió, tornam a la pàgina del clúster a Atlas i pitjam el botó "Connect". A continuació hem de seleccionar l'opció "Compass":



**Imatge:** Crear una connexió al clúster

Ara hem de seleccionar l'opció "I have MongoDB Compass installed" i deixar seleccionada que la versió és 1.38 o posterior. Ens mostra quina és la cadena que hem de copiar per poder connectar-nos des de Compass. En el meu cas, que estic treballant amb l'usuari *toni*, la cadena és: *mongodb+srv://toni:<db\_password>@cluster0.5c711.mongodb.net/*

## Connect to Cluster0

**Set up connection security**

**Choose a connection method**

**3 Connect**

**Connecting with MongoDB Compass**

I don't have MongoDB Compass installed

I have MongoDB Compass installed

**1. Choose your version of Compass**

1.38 or later

See your Compass version in “About Compass”

**2. Copy the connection string, then open MongoDB Compass**

Use this connection string in your application

```
mongodb+srv://toni:<db_password>@cluster0.5c7l1.mongodb.net/
```

Replace <db\_password> with the password for the toni user. Ensure any options are URL encoded.

**RESOURCES**

- Connect with Compass
- Import and Export Data
- Access your Database Users
- Troubleshoot Connections

**Go Back** **Done**

**Imatge:** Cadena de connexió per a Compass

Ara ja podem tancar la finestra d'Atlas, tornar a Compass i enganxar la cadena de connexió, substituint <password> per la contrasenya de l'usuari que hem creat anteriorment. Recordau que hem hagut d'habilitar la nostra IP des d'Atlas perquè ens permeti connectar-nos-hi.

## New Connection

Manage your connection settings

**URI**

mongodb+srv://toni:\*\*\*\*\*@cluster0.5c7l1.mongodb.net/

**Name** cluster0.5c7l1.mongodb.net

**Color** No Color

Favorite this connection

Favoriting a connection will pin it to the top of your list of connections

**Advanced Connection Options**

**Cancel** **Save** **Connect**

**How do I find my connection string in Atlas?**

If you have an Atlas cluster, go to the Cluster view. Click the ‘Connect’ button for the cluster to which you wish to connect.  
See example

**How do I format my connection string?**

See example

**Imatge:** Crear una nova connexió en Compass

Podem pitjar "Connect" per a crear la connexió.

Ara ja podem veure de manera gràfica les nostres bases de dades, col·leccions i documents. La següent imatge mostra les dades (documents) dels primers restaurants de la col·lecció restaurants, en format JSON (icona amb les claus):

```

[{"_id": "ObjectId('5eb3d668b31de5d588f4292a')", "address": {"object": "Brooklyn"}, "borough": "Brooklyn", "cuisine": "American", "grades": [{"array": 4}, {"name": "Misteria Caterers"}, {"restaurant_id": "40336018"}], {"_id": "ObjectId('5eb3d668b31de5d588f4292b')", "address": {"object": "Delicatessen"}, "borough": "Delicatessen", "cuisine": "American", "grades": [{"array": 4}, {"name": "Wilkens Fine Food"}, {"restaurant_id": "40336433"}], {"_id": "ObjectId('5eb3d668b31de5d588f4292c')", "address": {"object": "Kosher"}, "borough": "Kosher", "cuisine": "American", "grades": [{"array": 4}, {"name": "Kosher Island"}, {"restaurant_id": "40336442"}], {"_id": "ObjectId('5eb3d668b31de5d588f4292d')", "address": {"object": "Hamburgers"}, "borough": "Hamburgers", "cuisine": "American", "grades": [{"array": 4}, {"name": "Wendy's"}, {"restaurant_id": "30112349"}]}

```

**Imatge:** Col·lecció restaurants en Compass (vista JSON)

Igual que amb Atlas, podem emprar el camp "Filter" per definir filtres i el botó "Options" per establir, entre d'altres, l'ordenació de les dades. A més, podem executar comandes fent servir mongosh, el shell de MongoDB, que podem obrir pitjant el botó "Open MongoDB shell", a la part superior dreta de la pantalla.

**MongoDB Shell**, o **mongosh**, es un entorn JavaScript i Node.js, completament funcional, que ens permet interactuar amb els clústers de MongoDB.

Podem interactuar amb mongosh a través de Compass, mitjançant el botó "Open MongoDB shell".

També, però, podem fer-ho des d'un terminal del sistema operatiu. Per fer-ho, hem d'instal·lar l'aplicació MongoDB Shell, que podeu descarregar des de <https://www.mongodb.com/try/download/shell>. Seleccionau la versió més recent per al vostre sistema operatiu. En el cas de Windows, podeu triar si descarregar un ZIP, o més recomanable, directament l'instal·lador MSI. La darrera versió (setembre de 2024) és la 2.3.1. Podeu consultar la pàgina <https://www.mongodb.com/docs/mongodb-shell/install/> si teniu algun dubte referent a la instal·lació.

Ara hem de saber quina és la cadena de connexió per utilitzar des de mongosh. Per fer-ho, anam a la pàgina del nostre cluster en Atlas, pitjam el botó "Connect", seleccionam l'opció "Shell", i en la nova pantalla marcam "I have the MongoDB Shell installed" i "mongosh (2.0 or later)". Ens apareixerà la cadena de connexió que hem de copiar per poder-nos connectar des de MongoDB Shell. En el meu cas, amb l'usuari *toni*, és mongosh "mongodb+srv://cluster0.5c7l1.mongodb.net/" --apiVersion 1 --username toni

**Connect to Cluster0**

Set up connection security      Choose a connection method      Connect (3)

**I don't have the MongoDB Shell installed**      **I have the MongoDB Shell installed**

**1. Select your mongo shell version**

To check your Mongo shell version, run:  
`mongosh --version OR mongo --version`

`mongosh (2.0 or later) ▾`

**2. Run your connection string in your command line**

**Use this connection string in your application**

`mongosh "mongodb+srv://cluster0.5c7l1.mongodb.net/" --apiVersion 1 --username toni`

You will be prompted for the password for the Database User, **toni**. When entering your password, make sure all special characters are [URL encoded](#).

**RESOURCES**

[Add Data in the Shell](#)      [Access your Database Users](#)

[Troubleshoot Connections](#)

**Go Back**      **Done**

**Imatge:** Cadena de connexió per a MongoDB Shell

Ara ja podem obrir un terminal (Símbolo de sistema en Windows) i executar `mongosh` amb la vostra cadena de connexió al cluster, per entrar en el shell de MongoDB:

```
C:\toni>mongosh "mongodb+srv://cluster0.5c7l1.mongodb.net/" --apiVersion 1 --username toni
Enter password: *****
Current Mongosh Log ID: 66e9ba58a09e179d19c73bf7
Connecting to:      mongodb+srv://<credentials>@cluster0.5c7l1.mongodb.net/?appName=mongosh+2.3.1
Using MongoDB:     7.0.12 (API Version 1)
Using Mongosh:    2.3.1

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

Atlas atlas-l76dmh-shard-0 [primary] test> show dbs
sample_airbnb      52.10 MiB
sample_analytics   9.56 MiB
sample_geospatial  1.24 MiB
sample_guides      40.00 KiB
sample_mflix       112.88 MiB
sample_restaurants 6.39 MiB
sample_supplies    1.05 MiB
sample_training    50.59 MiB
sample_weatherdata 2.61 MiB
admin              328.00 KiB
local              1.13 GiB
Atlas atlas-l76dmh-shard-0 [primary] test> |
```

**Imatge:** MongoDB Shell en un terminal de Windows

NOTA: Aquesta comanda per obrir el shell amb el nostre clúster sempre serà la mateixa, així que no cal anar cada vegada a cercar-la a Atlas. Millor apuntar-la o fer un arxiu .bat que l'executi.

Podem veure quines bases de dades tenim al nostre clúster, executant la següent ordre:

```
show dbs
```

El resultat de l'execució és aquest:

sample_airbnb	52.10 MiB
sample_analytics	9.56 MiB
sample_geospatial	1.24 MiB
sample_guides	40.00 KiB
sample_mflix	112.88 MiB
sample_restaurants	6.39 MiB
sample_supplies	1.05 MiB
sample_training	50.59 MiB
sample_weatherdata	2.61 MiB
admin	328.00 KiB
local	1.13 GiB

Ara anam a establir que farem feina amb la base de dades sample\_restaurants:

```
use sample_restaurants
```

Ara voldríem saber quines col·leccions estan emmagatzemades a aquesta base de dades:

```
show collections
```

Ens retorna el següent resultat:

```
neighborhoods
```

```
restaurants
```

## 5.3. Interactuar amb les col·leccions

Una vegada que hem establert quina és la base de dades amb la qual treballam, tenim un objecte **db** per interactuar amb ella. Aquest objecte *db* conté un objecte per interactuar amb cada una de les col·leccions, amb el mateix nom de la col·lecció. Així doncs, **db.restaurants** fa referència a la col·lecció *restaurants* de la base de dades actual (*sample\_restaurants*). I ara, mitjançant aquest objecte de la col·lecció podem accedir a tots els mètodes per a fer les operacions CRUD. Per exemple, emprarem el mètode **find()**, que recupera les instàncies de la col·lecció. Així que l'ordre que escriurem en MongoSh seria:

```
db.restaurants.find()
```

En la imatge següent podem veure l'execució d'aquestes ordres en la pestanya MongoSh de Compass:

The screenshot shows the Compass application interface. On the left, there's a sidebar with 'My Queries' and a 'CONNECTIONS (2)' section containing 'Cluster0' and 'cluster0.5c711.mongodb.net'. The 'cluster0.5c711.mongodb.net' connection is expanded, showing databases like 'admin', 'config', 'local', and several sample databases including 'sample\_airbnb', 'sample\_analytics', 'sample\_geospatial', 'sample\_guides', 'sample\_mflix', 'sample\_neighborhoods', 'sample\_restaurants', 'sample\_supplies', 'sample\_training', and 'sample\_weatherdata'. The right side of the screen is the MongoSh terminal window. It displays the command 'db.restaurants.find()' followed by its execution result, which is a list of documents from the 'sample\_restaurants' collection. Each document includes fields like '\_id', 'address', 'building', 'coord', 'street', 'zipcode', 'borough', 'cuisine', 'grades', and 'date'.

```
> db.restaurants.find()
< [
  {
    "_id": ObjectId('5eb3d668b31de5d588f4292a'),
    "address": {
      "building": '2780',
      "coord": [
        -73.98241999999999,
        40.579505
      ],
      "street": 'Stillwell Avenue',
      "zipcode": '11224'
    },
    "borough": 'Brooklyn',
    "cuisine": 'American',
    "grades": [
      {
        "date": 2014-06-10T00:00:00.000Z,
        "grade": 'A',
        "score": 5
      },
      {
        "date": 2013-06-05T00:00:00.000Z,
        "grade": 'A',
        "score": 7
      },
      {
        "date": 2012-04-13T00:00:00.000Z,
        "grade": 'A',
        "score": 12
      }
    ],
    "name": "Vito's Authentic Italian",
    "rating": 4.5
  }
]
```

imatge: MongoSh en Compass

Si volem establir un filtre, li passarem per paràmetre al mètode *find* la condició que s'ha de cumplir. En l'exemple, que sigui de cuina espanyola:

```
db.restaurants.find( {"cuisine": "Spanish"} )
```

Lògicament, en els filtres també podem emprar tots els camps de l'arbre del document: camps dins camps que són un subdocument. Per exemple, un document *restaurant* té un camp *address*, que està format per un subdocument que té un camp *street*. Així doncs, el filtre per trobar els restaurants de la cinquena avinguda seria així:

```
{"address.street": "Fifth Avenue"}
```

Si s'han de cumplir diverses condicions, les separarem per comes. Per exemple, el filtre per als restaurants de cuina espanyola al districte de Brooklyn seria:

```
{"cuisine": "Spanish", "borough": "Brooklyn"}
```

L'objecte de la col·lecció també té un mètode **count()** (o **countDocuments()** en les darreres versions) que ens retorna el nombre d'instàncies de la col·lecció:

```
db.restaurants.count()
```

Això ens retorna 25359.

També podem aplicar el mètode **count()** al resultat d'una operació de *find*, per saber quantes instàncies retorna:

```
db.restaurants.find( {"cuisine": "Spanish"} ).count()
```

Que ens retornaria 637.

Podem també emprar el mètode **sort()** per ordenar una col·lecció o el resultat d'una operació *find*. Per fer-ho, especificam el camp pel qual volem ordenar i si volem emprar un ordre ascendent (1) o descendent (-1). Per exemple:

```
db.restaurants.find( {"cuisine": "Spanish"} ).sort({"name": -1})
```

retorna els restaurants de cuina espanyola, ordenats pel seu nom, en ordre descendent (invers).

El mètode **limit()** ens permet limitar el nombre d'instàncies que recuperam mitjançant una operació *find*. Li passam per paràmetre el nombre d'instàncies. Per exemple, la següent operació, una vegada ha recuperat els restaurants de cuina espanyola, ordenats en ordre invers, en recupera només una instància, la primera. És a dir, ens retornarà el restaurant amb el nom alfabèticament més gran: seria l'equivalent de fer un *max* de SQL sobre el camp *name*.

```
db.restaurants.find( {"cuisine": "Spanish"} ).sort({"name": -1}).limit(1)
```

Per últim, un altre mètode útil de l'objecte que representa una col·lecció és **distinct()**, que ens permet recuperar els valors diferents que té un determinat camp. Per exemple:

```
db.restaurants.distinct("borough")
```

Això ens retorna tots els valors que pren el camp *borough* a la col·lecció *restaurants*, és a dir, els barris o districtes de Nova York:

```
[  
  'Bronx',  
  'Brooklyn',  
  'Manhattan',  
  'Missing',  
  'Queens',  
  'Staten Island'  
]
```



Podeu trobar més detalls sobre els mètodes de l'objecte col·lecció a la documentació de MongoDB:

<https://www.mongodb.com/docs/manual/reference/method/js-collection/>

I també sobre l'objecte cursor (el resultat d'una operació de find):

<https://www.mongodb.com/docs/manual/reference/method/js-cursor/>

AMPLIACIÓ

## 5.4. Operacions CRUD

Recordem que les bases de dades NoSQL no treballen amb SQL (com el seu nom indica) i que la manera d'accendir a les dades és a través d'una API específica per a cada gestor. Així, quan desenvolupem una aplicació, per exemple per treballar amb MongoDB, haurem de fer feina amb els objectes d'aquesta API. És per això que és important saber com s'implementen les operacions de creació, lectura, modificació i esborrat de documents (operacions CRUD), mitjançant l'API de MongoDB.

En aquest mòdul no desenvoluparem cap aplicació, però sí treballarem amb els objectes de l'API a través de mongosh, el shell de MongoDB.

### Creació

El que volem fer és inserir un document en una col·lecció. El mètode que utilitzarem és `insertOne()`. Li hem de passar per paràmetre el document JSON de la instància que volem inserir a la col·lecció.

El següent exemple insereix un nou restaurant a la col·lecció restaurants, amb el nom "Pa-ella", de cuina espanyola, al barri de Brooklyn:

```
db.restaurants.insertOne(
  { "borough": "Brooklyn", "cuisine": "Spanish", "name": "Pa-ella" }
)
```

L'identificador del nou restaurant (`_id`) es genera automàticament.



Les cometes en els noms dels camps no són obligatòries, ni en aquest exemple, ni en els següents. Podríem escriure-ho també així:

```
db.restaurants.insertOne({
  borough: "Brooklyn",
  cuisine: "Spanish",
  name: "Pa-ella"
})
```

ALERTA

Aquí podem veure la seva execució en mongosh, que ens diu que l'operació ha estat vàlida i ens mostra l'identificador generat:

```
> db.restaurants.insertOne( { "borough": "Brooklyn", "cuisine": "Spanish", "name": "Pa-ella" } )
< { acknowledged: true,
  insertedId: ObjectId("63bb76144fedd1a8c71a247") }
```

**Imatge:** Inserció d'un document

Opcionalment, podem especificar en quants de nodes s'ha d'escriure el nou document perquè l'operació es consideri vàlida, mitjançant un `writeConcern` (un altre document JSON). Vegem un exemple:

```
db.restaurants.insertOne(
  { "borough": "Brooklyn", "cuisine": "Spanish", "name": "Pa-ella" },
  { writeConcern: {w: "majority", wtimeout:5000} }
)
```

El camp w: "majority" indica que s'ha d'escriure a la majoria dels nodes; quants més nodes, més consistència, però més temps. Des de MongoDB 5.0, el valor per defecte és "majority" (abans era 1).

El camp wtimeout indica el temps límit, que si se sobrepassa sense haver fet les insercions corresponents, es considerarà una operació invàlida.

Tenim també l'opció d'inserir diverses instàncies a la vegada, fent servir el mètode **insertMany()**. En aquest cas, el document JSON haurà de contenir els documents de totes les instàncies que volem inserir.



Si afegim un document (una instància) a una col·lecció que no existeix, automàticament es crea una nova col·lecció dins la base de dades actual. Per exemple:

```
db.persones.insertOne( {"nom": "Toni", "edat": 49, "altura": 186} )
```

**IMPORTANT**

Crea una nova col·lecció *persones* i li insereix una nova instància, amb els camps nom (Toni), edat (49) i altura (186).

## Modificació

Tenim dos mètodes per modificar documents existents en una col·lecció: **updateMany()** i **updateOne()**.

El paràmetre d'aquests mètodes té dos documents:

- un filtre, que han de satisfet el(s) document(s) que volem modificar
- quin canvi volem fer, especificant una clàusula \$set

Vegem un exemple, on volem modificar el barri del restaurant que hem inserit abans.

```
db.restaurants.updateOne(
    { "name" : "Pa-ella" },
    { $set: { "borough": "Manhattan" } }
)
```

Podem veure que, en el primer lloc especificam el filtre, que en aquest cas, és que el nom sigui "Pa-ella":

```
{ "name" : "Pa-ella"}
```

A continuació, especificam la modificació que volem fer, en aquest cas, canviar el barri a Manhattan:

```
{ $set: { "borough": "Manhattan" } }
```

Aquí hem emprat el mètode updateOne(), perquè tanmateix, sabíem que només hi ha un restaurant amb aquest nom. Però si n'hi pogués haver més d'un i volguéssim modificar-los tots de cop, podríem emprar el mètode updateMany(). En canvi, emprant updateOne() modifica el primer que troba que satisfà el filtre, i ja no en cerca més.

Aquí podem veure l'execució en mongosh:

```
> db.restaurants.updateOne(
    { "name" : "Pa-ella" },
    { $set: {"borough": "Manhattan"} }
)
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 0,
    upsertedCount: 0 }
```

Imatge: Modificació d'un document

Podem especificar algunes propietats més relacionades amb l'operació. Una és el *writeConcern* que hem vist abans. Una altra és *upsert*, que és un booleà que si es posa a true, indica que si no es troba cap instància que satisfaci el filtre, en creï una de nova.

## Esborrat

Tenim dos mètodes per eliminar documents existents en una col·lecció: **deleteMany()** i **deleteOne()**. Igual que hem vist amb les modificacions, el primer esborra totes les instàncies que satisfan el filtre, mentre que el segon només n'esborra la primera que troba.

En aquest cas, com a paràmetre hem d'especificar un document amb el filtre que s'ha de satisfer.

El següent exemple, elimina el restaurant amb nom "Pa-ella":

```
db.restaurants.deleteOne(
    { "name" : "Pa-ella" }
)
```

Vegem l'execució en mongosh:

```
> db.restaurants.deleteOne(
    { "name" : "Pa-ella" }
)
< { acknowledged: true, deletedCount: 1 }
```

Imatge: Esborrat d'un document

També aquí podem especificar altres propietats de l'operació, com el *writeConcern*.

## Lectura (recerca)

En l'apartat anterior ja vàrem veure el mètode **find()**, que recupera els documents d'una col·lecció que satisfan una condició específica (un filtre). Recordem que com a paràmetre li passam un document especificant el filtre.

També tenim un mètode equivalent, **findOne()**, que recupera només el primer document que troba que satisfà el filtre.

A més, tant en el cas de `find()` com de `findOne()`, podem especificar quins camps volem recuperar (per defecte, tots). És el que s'anomena una projecció.

Vegem un exemple que recupera tots els restaurants del barri Queens:

```
db.restaurants.find(
    { "borough" : "Queens" }
)
```

Això ens retorna un document que conté els 5.656 restaurants de Queens.

Si només volguéssim recuperar el nom i el tipus de cuina d'aquests restaurants, ho faríem especificant els camps de la projecció (*name* i *cuisine*):

```
db.restaurants.find(  
  { "borough" : "Queens"},  
  { "name":1, "cuisine":1}  
)
```

En lloc de 1, podem posar true.



Podeu trobar més detalls sobre les operacions CRUD a la documentació de MongoDB:

<https://www.mongodb.com/docs/manual/crud/>

AMPLIACIÓ

## 5.5. Operadors

### Operadors de comparació

En les expressions dels filtres, tant en els mètodes de find com de update o delete, podem emprar diversos operadors per comparar el valor del camp amb un valor determinat:

- **\$eq (equal)**: existeix i és igual que el valor
- **\$ne (not equal)**: no existeix o, si existeix, no és igual que el valor
- **\$lt (less than)**: existeix i és menor que el valor
- **\$lte (less than or equal)**: existeix i és menor o igual que el valor
- **\$gt (greater than)**: existeix i és major que el valor
- **\$gte (greater than or equal)**: existeix i és major o igual que el valor
- **\$in**: existeix i el valor està dins de l'array
- **\$nin**: no existeix o, si existeix, el valor no està dins de l'array

Per exemple, sobre una col·lecció personnes, el filtre:

```
{ edat: {$gte: 18 } }
```

recupera totes les instàncies que tenen un camp *edad* i que els seu valor és 18 o major.

```
{ altura: {$gt: 180, $lt: 190 } }
```

recupera totes les instàncies que tenen un camp *altura* i que el seu valor és major que 180 i menor que 190 (sense incloure ni 180 ni 190). S'han de cumplir les dues condicions.

---

```
{ fills: {$ne: 0 } }
```

recupera totes les instàncies que no tenen un camp *fills*, o que si el tenen, el seu valor no és 0.

En el cas dels arrays, suposem que en l'exemple dels restaurants, poguéssim tenir restaurants amb diversos tipus de cuina, com en el següent cas:

```
{
  "borough": "Brooklyn",
  "cuisine": ["Spanish", "American"],
  "name": "Pa-ella"
}
```

Si volguéssim que només ens retornàs els restaurants que només siguin de cuina "Spanish", sense incloure d'altres (no ens retornaria el "Pa-ella"), hauríem d'indicar l'array exacte que ha de tenir el camp *cuisine*:

```
{ "cuisine" : ["Spanish"] }
```

Però si volguéssim que retornàs tots els que tenguin "Spanish", encara que també puguin tenir altres (sí ens retornaria el "Pa-ella"), posaríem el valor que ha de contenir (no tot l'array). Seria així:

```
{ "cuisine" : "Spanish" }
```

També podríem emprar l'operador *\$in*. Si algun dels valor que conté l'array de la instància està dins l'array que li passam a l'operador *\$in*, ens retornarà la instància. Així, { "cuisine" : "Spanish" } ho podríem escriure també així:

```
{ "cuisine" : { $in: ["Spanish"] } }
```

Això ens permetria, a més, escriure una consulta que retorne els que tenguin una cuina que sigui espanyola, portuguesa o italiana (és a dir, que el seu valor estigui dins l'array ["Spanish", "Portuguese", "Italian"]):

```
{ "cuisine" : { $in: ["Spanish", "Portuguese", "Italian"] } }
```

Fixau-vos que potser que a més d'alguna d'aquestes tres, en tenguin d'altres. I que també que en tenguin més d'una d'aquestes tres. El que sabem cert és que tots els que recuperi tendran almenys una de les tres.

## Operador d'existència

Recordau que, a diferència d'una taula d'una base de dades relacional, no totes les instàncies d'una col·lecció tenen els mateixos camps. L'operador `$exists` ens permet recuperar les instàncies que contenguin un determinat camp. Per entendre-ho, imaginem que els restaurants amb una estrella Michelin tenen un camp `michelinStars`, el valor del qual indica quantes estrelles té. Suposem que el nostre Pa-ella té una estrella Michelin:

```
{
  "borough": "Brooklyn",
  "cuisine": ["Spanish", "American"],
  "name": "Pa-ella",
  "michelinStars": 1
}
```

Si volguéssim recuperar els restaurants amb el camp `michelinStars` ho faríem emprant l'operador `$exists`, així:

```
{"michelinStars": {$exists:true}}
```

## Operadors lògics

Abans hem vist una query que tenia dues subcondicions: que l'altura fos major que 180 i que també fos menor que 190. Perquè la condició global sigui certa, s'han de cumplir les dues. Però podem emprar altres operadors lògics per definir els nostres filtres:

- **\$and** (i lògica): s'han de cumplir les dues subcondicions perquè la condició global sigui certa. És l'equivalent a l'exemple de l'altura que hem vist abans. Per exemple:

```
db.restaurants.find({
  $and: [{ "borough" : "Bronx"}, {"cuisine": "Chinese"} ]
})
```

Retorna tots els restaurants de cuina xinesa del Bronx: han de passar les dues subcondicions, és a dir, que la cuina sigui xinesa i que el barri sigui el Bronx. Si només passa una de les dues, no el retornarà.

- **\$or** (o lògica): basta que es compleixi una de les subcondicions perquè la condició global sigui certa. Si les dues són certes, la condició global també és certa. Només serà fals si les dues subcondicions són falses. Per exemple:

```
db.restaurants.find({
  $or: [{ "borough" : "Bronx"}, {"cuisine": "Chinese"} ]
})
```

En aquest cas, retorna els restaurants que estiguin al Bronx, o que siguin de cuina xinesa: basta que se compleixi una de les dues subcondicions (o les dues).

- **\$not** (negació): és la negació d'una subcondició. És a dir, si la subcondició és certa, la condició global serà falsa, i viceversa. Per exemple:

```
db.restaurants.find({ "borough": {$not: {$eq: "Bronx"}} })
```

Recupera els restaurants que no són del barri del Bronx. En aquest cas, hagués estat més fàcil escriure-ho amb l'operador `$ne`.

- **\$nor:** es la negació d'una o lògica. És a dir, només serà certa si totes les subcondicions són falses (si una d'elles és certa, la condició global serà falsa). Es fa servir molt menys que les anteriors.

## Expressions regulars

L'operador `$regex` ens permet utilitzar expressions regulars en les queries. Podeu trobar informació sobre què són i com es construeixen expressions regulars a moltes fonts, per exemple a la Wikipedia: [https://ca.wikipedia.org/wiki/Expressió\\_regular](https://ca.wikipedia.org/wiki/Expressió_regular)

La següent query recupera tots els restaurants (només un en aquesta col·lecció) que el seu nom conté el text "pizza":

```
db.restaurants.find( {"name": {$regex:/pizza/}} )
```

La forma genèrica d'un filtre amb una expressió regular és la següent:

```
{ <camp>: { $regex: /<patró>/<opcions> } }
```

Entre les opcions, la més habituals és `i`, que fa que la cerca sigui *case insensitive*, és a dir, que no distingeix entre majúscules i minúscules. Per exemple, la següent query cerca de nou "pizza", però amb aquesta opció *case insensitive*, amb la qual cosa, ara retorna 1.261 restaurants.

```
db.restaurants.find( {"name": {$regex:/pizza/i}} )
```

Com és lògic, podem utilitzar patrons més complexos en la nostra expressió regular. Per exemple, la següent query fa servir l'àncora `^` per recuperar els restaurants que comencen per "pizza" (case insensitive)

```
db.restaurants.find( {"name": {$regex:/^pizza/i}} )
```

mentre que la següent empra l'àncora `$` per recuperar els que acaben per "pizza":

```
db.restaurants.find( {"name": {$regex:/pizza$/i}} )
```

Per acabar amb aquesta secció, la següent query recupera tots els restaurants amb un nom que comença per un número:

```
db.restaurants.find( {"name": {$regex:/^[0-9]/}} )
```

## Operadors de modificació de camps

En l'apartat anterior hem vist que en l'operació de modificació `updateOne()` hem emprat un operador `$set` per assignar el nou valor al camp `borough`:

```
db.restaurants.updateOne(
    { "name" : "Pa-ella" },
    { $set: {"borough": "Manhattan"} }
)
```

Però, a més de \$set, podem emprar altres operadors:

- **\$currentDate**: assigna la data actual
- **\$inc**: incrementa el valor
- **\$mul**: multiplica el valor del camp
- **\$min**: modifica si valor especificat mitjançant \$min és menor que l'actual
- **\$max**: modifica si valor especificat mitjançant \$max és major que l'actual
- **\$unset**: elimina el camp de la instància

Si tenim la col·lecció persones que hem creat abans amb l'operació:

```
db.persones.insertOne( { "nom": "Toni", "edat": 49, altura: 186 }
```

La següent operació incrementarà en 1 el camp edat de la instància Toni (passarà a valer 50):

```
db.persones.updateOne( { "nom": "Toni" }, { $inc: { "edat": 1 } } )
```

Si ara executam la següent operació de modificació, no tindrà cap efecte, perquè el valor del camp edat (50) era menor que el que li hem especificat amb l'operador \$min (65).

```
db.persones.updateOne( { "nom": "Toni" }, { $min: { "edat": 65 } } )
```



Podeu trobar més detalls sobre els operadors de comparació a la documentació de MongoDB:

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

Sobre l'operador \$exists:

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

Sobre els operadors lògics:

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

Sobre l'operador \$regex:

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

I sobre els operadors de modificació:

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