

Maestría en Análisis y Visualización de Datos Masivos

Bases de Datos para Datos Masivos

Bases de Datos para el Big Data

Tema 1. Métodos de captura de información

Índice

[Esquema](#)

[Ideas clave](#)

- [1.1. Introducción y objetivos](#)
- [1.2. Origen y calidad de los datos](#)
- [1.3. Organización de los datos](#)
- [1.4. Casos de estudio](#)
- [1.5. Referencias bibliográficas](#)

[A fondo](#)

- [How do scientists collect data?](#)
- [Introduction to JSON data](#)
- [The relational model](#)
- [Well-formed XML](#)
- [Introducción a SQL](#)
- [Introducción a MySQL](#)
- [7 command-line tools for data science](#)
- [SQL for Web Nerds](#)
- [Dataportals.org](#)
- [Research-Quality Data Sets](#)
- [JSON.org](#)
- [Bibliografía](#)

[Test](#)

MÉTODOS DE CAPTURA DE INFORMACIÓN		
Origen y calidad de los datos	Organización de los datos	Casos de estudio
<p>Evaluación de calidad:</p> <ul style="list-style-type: none"> - Complejitud: grado en el que los valores se encuentran en un conjunto de datos. - Credibilidad: nivel de fiabilidad del organismo que proporciona el conjunto de datos. - Consistencia: grado en el que los datos carecen de contradicciones. - Interpretabilidad: grado en el que los datos deben ser interpretados por una persona. - Precisión: nivel de exactitud del valor. 	<p>Ficheros planos:</p> <ul style="list-style-type: none"> - CSV (comma separated values): RFC 4180. Define un registro por línea y separa los campos por comas. - JSON: RFC 7159 y ECMA-404. Describe objetos encapsulados por llaves y listas por corchetes. - XML: descrito en el estándar XML 1.0 de W3C. Permite almacenar información de forma legible utilizando etiquetas. <p>Bases de datos:</p> <ul style="list-style-type: none"> - Conjunto de datos persistentes, utilizados por sistemas de aplicación. - En el modelo Entidad-Relación (E/R) una entidad es cualquier objeto representado en la BBDD y un vínculo representa relaciones entre ellos. - La unidad básica de almacenamiento es el campo, agrupados en registros y estos en ficheros almacenados. 	<p>Procesamiento de sitio web sobre cursos online.</p> <p>Procesamiento de logs de servidor web.</p> <p>API de acceso a transacciones bancarias.</p> <p>Almacenamiento de información sobre productos en un fichero CSV.</p> <p>Representación de información geolocalizada en formato JSON.</p> <p>Almacenamiento de información sobre clientes de una base de datos relacional.</p>
<p>Niveles de abstracción:</p> <ul style="list-style-type: none"> - Datos: conjunto de hechos discretos y objetivos sobre un evento. - Información: datos con significado. - Conocimiento: combinación de información contextualizada, experiencias, valores e intuición. 		<p>Bases de datos y SQL:</p> <ul style="list-style-type: none"> - Los ficheros almacenados se representan en forma de tablas (relaciones), con columnas (campos) y filas (registros). - El estándar SQL define un lenguaje para la consulta y modificación de los datos, - Los comandos INSERT, UPDATE y DELETE permiten la inserción, edición y eliminación de registros respectivamente.
<p>Fuentes de datos:</p> <ul style="list-style-type: none"> - Captura manual: encuestas y observaciones. - Análisis de documentos estructurados: estructurados (HTML) y sin formato (lenguaje natural). - Salida de aplicaciones: logs o bases de datos. - Sensores: dispositivos de medición. - Datos de acceso público: gubernamentales y servicios web públicos. 		

1.1. Introducción y objetivos

Este tema realiza una **introducción a la asignatura** y presenta las **definiciones necesarias** sobre las que se basarán los temas posteriores. Para estudiar este tema se deben **leer las Ideas clave** y, si se desea información adicional sobre un concepto específico, se debería consultar las **lecturas sugeridas en la bibliografía** (sección A fondo).

La lectura de los casos de estudio también será de ayuda, ya que proporcionan información de las decisiones tomadas para la captura de datos en un entorno determinado. Este capítulo servirá tanto de introducción como de repaso de los conceptos básicos sobre bases de datos en general y bases de datos relacionales en específico.

En tiempos pasados, la obtención de datos solía ser un proceso costoso y a menudo requería del trabajo humano para la digitalización de documentos físicos. El continuo avance de la tecnología permite que los métodos disponibles para la captura de datos sean cada vez **más diversos**. Esto, en adición a la disminución del **coste por unidad de almacenamiento**, ha resultado en la posibilidad de almacenar grandes cantidades de datos provenientes de distintas fuentes.

A raíz de esta diversidad, es necesario contar con tres elementos que conforman la base para capturar datos de forma eficiente y adecuada al uso que se le brindará a dichos datos:

- ▶ El primer elemento es la definición y distinción de los términos *dato, información y conocimiento*.
- ▶ El segundo, las dimensiones o criterios de calidad que se analizan en un conjunto de datos.

- ▶ El último elemento es la experiencia de analizar casos verdaderos en los que ha sido necesario capturar datos y almacenarlos.

El aprendizaje de formatos de ficheros de texto como **CSV** y **JSON** será de utilidad en los próximos temas, ya que son formatos utilizados por el sistema de almacenamiento.

Después de capturar información y poder utilizarla de forma eficiente, será necesario almacenarla de **forma permanente**. Así, la información obtenida no residirá únicamente en la memoria volátil del ordenador, sino que estará disponible para futuras ocasiones.

El método más básico para almacenar datos es mediante el uso del **sistema de ficheros** del sistema operativo. Los ficheros pueden tener un **formato plano**, donde toda la información es legible para una persona; o un **formato binario**, donde la información puede escribirse y leerse de forma directa por una aplicación, pero no puede ser analizada directamente de forma manual.

Este tema proporciona un **repaso** de los mecanismos comúnmente utilizados para el **almacenamiento de información**. Se hablará de dos aproximaciones: la utilización de ficheros planos, que son comúnmente utilizados para el almacenamiento y compartición de datos, y las bases de datos, que dan un paso más allá y proporcionan una **consistencia en la información** y el hecho de poder **consultar y modificar de manera eficiente** un conjunto de datos en específico. Es por esto por lo que se revisará el concepto de base de datos y, en particular, las **bases de datos relacionales**.

Además de ser una de las herramientas más comunes en la actualidad, se puede hacer una analogía entre estas herramientas y el sistema que se utilizarán en otros temas de la asignatura.

1.2. Origen y calidad de los datos

Datos, información y conocimiento

En situaciones informales es común usar indiscriminadamente los términos **dato**, **información** y **conocimiento**. En ámbitos profesionales y académicos, es conveniente distinguir estos conceptos para evitar malinterpretaciones durante las distintas fases de la analítica de datos.

Existen varias aproximaciones para la distinción de estos términos. En el contexto de esta asignatura se utilizará la definición descrita por Davenport y Prusak (2000).

Un dato puede definirse como un hecho concreto y discreto acerca de un evento. La característica de ser discreto significa que, **semánticamente, es la unidad mínima que puede comunicarse o almacenarse**. Por sí solos, los datos no brindan detalles significantes del entorno del que fueron obtenidos. Ejemplos de datos pueden ser:

- ▶ 2010.
- ▶ 443.
- ▶ DE.

La **información** puede distinguirse simplemente como un **mensaje** formado por la **composición de varios datos**. Esto significa que, a diferencia del dato, la información sí posee un significado para un receptor u observador. Por ejemplo, utilizando los datos anteriores se podría obtener la siguiente información:

- ▶ El año de establecimiento de la empresa ACME fue el 2010.
- ▶ La altura del edificio Empire State es de 443 metros.
- ▶ DE es el código ISO que identifica al idioma alemán.

Los datos deben ser transformados para añadirles valor y convertirlos en información. Estas transformaciones incluyen métodos como:

- ▶ **Contextualización:** conocer el propósito del dato obtenido.
- ▶ **Categorización:** conocer la unidad de medida y los componentes del dato.
- ▶ **Cálculo:** realizar una operación matemática sobre el dato.
- ▶ **Corrección:** eliminar errores del dato.
- ▶ **Agregación:** resumir o minimizar un dato de forma más concisa.

El conocimiento implica una combinación de experiencias, información contextual y relevancia sobre cierta información. Así como la información se genera a partir de datos, el conocimiento surge de la agregación de información. Ejemplos de métodos que generan esta transformación son:

- ▶ **Comparación:** relación entre información obtenida en distintas experiencias.
- ▶ **Repercusión:** implicación de la información en decisiones y acciones.
- ▶ **Conexión:** relación entre distintos tipos de información.
- ▶ **Conversación:** opinión de otras personas sobre la información.

La jerarquía del conocimiento suele representarse gráficamente por una pirámide, siendo los datos la base y el conocimiento la cima.



Figura 1. Jerarquía del conocimiento.

Distinguir estos conceptos básicos proporciona un nivel de abstracción útil para la separación de características en el proceso de análisis. El hecho de que un dato sea inválido o erróneo **debe distinguirse fácilmente** de que la **información que se obtiene** de dicho conjunto de datos **sea adecuada o no** al problema que se intenta resolver.

Por ejemplo, es importante conocer si la malinterpretación de un análisis de datos se debe a un error en la fuente de datos, a un problema al combinar los datos en el proceso de análisis o a una confusión por parte del usuario final, debido a experiencias en otros contextos.

Evaluación de la calidad

Las métricas o dimensiones utilizadas para describir la calidad de un conjunto de datos pueden agruparse con base en los actores que interactúan con los datos. Los **diseñadores y administradores** de almacenes de datos tratan con métricas que afectan el diseño o esquema de los datos obtenidos y no con los datos directamente. Entre estas métricas se podría mencionar la *completitud* de los datos obtenidos y el minimalismo del conjunto de datos, el cual se interpreta como la eliminación de redundancias en el diseño del almacén de datos.

Los **desarrolladores de software** tratan con métricas específicas de la producción de productos informáticos. Si bien en esta categoría entran métricas que no están relacionadas con los datos directamente, es importante tener en cuenta que estas métricas afectan al proceso de almacenaje, acceso y manipulación de los datos.

El último actor que interactúa con los datos es el **usuario final**, es decir, quien utilizará los datos presentados para crear una conclusión y tomar una decisión acorde. Las métricas relevantes para este actor pueden ser el nivel de disponibilidad de los datos y el nivel de interpretación requerido para entender los datos presentados.

A partir de estas perspectivas, Jarke et al. (1998) presentan un conjunto de dimensiones para evaluar las propiedades de un conjunto de datos:

- ▶ La **completitud** o cobertura describe el porcentaje de datos disponibles respecto a la población total que representa dichos datos. Por ejemplo, un conjunto de datos con información de 90 de 100 tratamientos médicos realizados en un hospital presenta una cobertura de 90 %. Esto también se aplica a subconjuntos de los datos obtenidos, por ejemplo, el 5 % de los tratamientos médicos no incluyen la fecha de finalización del tratamiento, lo cual disminuiría la cobertura debido a esta característica.
- ▶ La **credibilidad** representa la fiabilidad que se le brinda al organismo que proporciona el conjunto de datos. Esta métrica puede reflejarse en el conjunto de datos por aquellas características que presenten un valor por defecto. Por ejemplo, si se habla de postres, es fácil ver que, si incluye valores por defecto como tarta, helado, fruta, etc., el dato es fiable, pero nunca lo será si se habla de pincho moruno.
- ▶ La **precisión** indica el porcentaje de datos correctos respecto al total disponible. También se representa por medio de un porcentaje.
- ▶ La **consistencia** describe el nivel con el que los datos son coherentes entre ellos. Un ejemplo de la aplicación de esta métrica se da en datos geográficos: se puede concluir que si una misma entidad tiene asociada una ciudad y un país que no tienen relación, existe un problema de consistencia.
- ▶ La **interpretabilidad** define el grado en el que los datos pueden ser entendidos correctamente por una persona. Entre los atributos que definen la interpretabilidad de un conjunto de datos se puede mencionar la documentación de elementos importantes y si el formato en el que se proporcionan los datos es entendible.

A continuación, se proporciona un conjunto de datos de fuente desconocida representadas en la siguiente tabla:

Datos de ejemplo					
Marca	Modelo	Precio	Motor	Consumo	Fecha
Audi	La Ferrari	15 000 €	963 CV diésel	alto	2014
Lamborghini	Murciélagos	300 000 €	580 CV 7500 rpm		2011
BMW	M6	158 000€	560 CV 4935 rpm	13,6/7,6/9,9	2012
Porsche	911 turbo	127 000 €	420 CV 7500 rpm	11 litros	2017

Tabla 1. Datos de ejemplo.

Si analizamos estos datos con respecto a los indicadores de calidad antes mencionados:

- ▶ **Compleitud:** 75 % respecto a los datos de consumo, o el mismo porcentaje con respecto a las revoluciones por minuto del motor del coche.
- ▶ **Credibilidad:** al tratarse de una fuente desconocida, la credibilidad es baja debido a que no se puede comprobar la fuente. Además, por defecto, se sabe que Ferrari solo hace motores de gasolina y que, por defecto, no existen motores diésel de ese caballaje.
- ▶ **Precisión:** es claro ver que los datos no son precisos, ya que el precio de un súper deportivo no puede ser de 15 000€ y las revoluciones de algunos coches son demasiado redondas. Además, un consumo «alto» no es una cifra de consumo.
- ▶ **Consistencia:** la consistencia de los datos es baja porque a poco que se entienda algo de coches, el modelo de Ferrari nunca puede pertenecer a la marca Audi, sino a Ferrari.

- ▶ **Interpretabilidad:** podría mejorarse, puesto que hay muchos datos que no incluyen unidades o que ofrecen un conjunto de números que si no se está familiarizado con el entorno no se conocerá su significado, como ocurre con los 3 valores de consumo.

Fuentes de información

Los métodos para la captura de información se pueden clasificar con base en las características del elemento que genera el conjunto de datos. Las cinco categorías más utilizadas en la actualidad son:

- ▶ La **captura manual** de datos es la categoría más tradicional y también una de las más frecuentes en el contexto de investigación en ámbitos sociales y naturales. Entre los métodos que encajan en esta categoría se encuentra el uso de encuestas y las mediciones a través de **observaciones**.

Si bien esta es la única categoría que no tiene una dependencia directa de las tecnologías de información, para su utilización es necesario que la información se digitalice, ya sea en el momento de la captura o en un procesado posterior.

- ▶ El **procesado de documentos estructurados** consiste en la obtención directa de datos disponibles en documentos, cuyo fin inicial no es ser consultados como fuente de datos. Uno de los métodos más comunes en esta categoría es el procesado de páginas HTML en un sitio web, conocido como **web scraping**.

Otro ejemplo es el análisis de **logs** o ficheros que contienen un listado secuencial de los eventos ocurridos dentro de un sistema y son creados con el objetivo de tener una bitácora y no para ser accedido por otras aplicaciones.

- ▶ La **salida de aplicaciones** es una de las categorías más triviales. Los métodos de este tipo involucran el acceso a almacenes de datos tradicionales, tales como **bases de datos relacionales**, ficheros con valores separados por comas (**CSV**), etc.
- ▶ Los **datos obtenidos a través de sensores**. En este conjunto se pueden mencionar ejemplos como sensores meteorológicos, sensores de ambiente (ruido o luz),

sensores corporales (ritmo cardíaco o conductividad de la piel) y sensores de dispositivos móviles (acelerómetro o giroscopio).

Actualmente existe un gran interés en el uso de sensores para la captura de datos en el contexto personal; este movimiento es conocido como ***quantified self***.

- ▶ El **acceso a datos públicos**, ya sea mediante la descarga de conjuntos de datos o a través de interfaces de programación de aplicaciones (**API**, por sus siglas en inglés). Por un lado, muchas de las entidades públicas (como gobiernos centrales y locales) publican catálogos de datos para que sean analizados y se desarrollen nuevas aplicaciones a partir de ellos.

La publicación de estos datos de carácter público suele realizarse en un portal dedicado, por ejemplo <http://datos.gob.es> en España, <http://data.gov.uk> en Reino Unido y <http://data.gov> en Estados Unidos. Por otro lado, algunas empresas brindan acceso a datos de forma pública, siendo uno de los fines el formar parte de una plataforma de desarrollo, como, por ejemplo, los servicios de redes sociales (Facebook o Twitter) que suelen brindar un API tanto para obtener información sobre un usuario en particular, como para alterar dicha información.

1.3. Organización de los datos

Ficheros planos

Los **ficheros planos** suelen ser un mecanismo utilizado para el intercambio de información entre sistemas. Una de sus ventajas es que es posible ver y editar el contenido del fichero con una herramienta de edición de texto.

Estos ficheros suelen ser mucho más verbosos que los ficheros en formato binario, lo cual implica que su tamaño en el sistema de ficheros será mayor, así como las operaciones necesarias para procesar el contenido desde un programa de *software*. Entre los formatos de fichero plano más comunes se pueden mencionar CSV, JSON y XML. A continuación, se describen los detalles básicos de cada uno de ellos.

El formato **CSV** (*Comma Separated Values* o valores separados por coma), se documenta en la RFC 4180 y presenta las siguientes características:

- ▶ Cada registro se delimita por un **cambio de línea** (combinación de dos caracteres: CR y LF).
- ▶ Como su nombre indica, los valores de cada registro se separan mediante el **uso de comas**. Es requerido que el número de valores sea constante para todos los registros disponibles en el fichero.
- ▶ Los valores pueden estar encapsulados con **comillas dobles**. Esto es obligatorio en aquellos casos donde el valor incluye un cambio de línea, una coma o comillas dobles.
- ▶ Si un valor contiene comillas dobles, estas deben escaparse precediéndolas con otro carácter de comillas dobles. Por ejemplo: «Encargado de ""Business Model""»
- ▶ Opcionalmente, puede incluir una primera línea con los **nombres de los campos** que se incluyen en el fichero.

A continuación, se muestra un ejemplo de datos en formato CSV. Se puede apreciar que el fichero tiene 4 registros, con 3 valores cada uno. Este ejemplo incluye los títulos de cada valor en la primera fila.

Nombre, Edad, Cargo
Juan, 45, Director
Antonio, 35, "Gestor de
proyectos"
Pablo, 34, "Analista"
Pedro, 32, "Administrador de bases de
datos"

El siguiente formato, **JSON** (*JavaScript Object Notation* o notación de objetos en JavaScript) se basa en el lenguaje de programación JavaScript y organiza su notación en dos estructuras:

- ▶ Un objeto o registro, definido como un conjunto de pares nombre/valor.
- ▶ Un *array* o lista ordenada de valores.

Al ser estructuras de datos muy genéricas, suelen estar soportadas por la mayoría de los lenguajes de programación modernos. La definición de las estructuras sigue las siguientes condiciones:

- ▶ Un objeto se delimita por llaves ({ }) y los pares nombre/valor se separan por medio de comas, y entre el nombre y el valor se coloca el carácter dos puntos (:).
- ▶ Un *array* se delimita por corchetes ([]) y los valores se separan por comas.
- ▶ Cada valor, tanto dentro de un objeto como de un *array*, puede ser una cadena de texto delimitada por comillas dobles, un número, un valor booleano (*true/false*), el valor nulo (*null*), un objeto o un *array*.

La representación de los primeros dos registros del ejemplo en CSV sería la siguiente en formato JSON:

```
[{ "Nombre": "Juan", "Edad": 45, "Cargo": "Director" }, { "Nombre": "Antonio", "Edad": 35, "Cargo": "Gestor de proyectos" }]
```

Puede observarse que el formato JSON es más verboso, lo cual permite que los objetos en un documento contengan información heterogénea, al contrario que en un documento CSV.

Finalmente, el formato **XML** (*eXtended Markup Language* o lenguaje de marcas extensible) es el más verboso de los tres que se representan en este tema. Como su nombre lo indica, utiliza marcas o **etiquetas** como parte de la estructura y formato de los datos que contiene.

Entre las condiciones que debe cumplir un documento que siga el formato XML se encuentran las siguientes:

- ▶ El documento inicia con la línea: <?xml version="1.0"?>.
- ▶ Un documento XML tiene solamente un elemento raíz.
- ▶ Un elemento en XML se abre mediante una etiqueta delimitada por los signos menor que y mayor que (<etiqueta>) y se cierra con la misma etiqueta, pero incluyendo una barra (/) inmediatamente después del menor que (</etiqueta>).

- ▶ Los elementos pueden tener atributos. Estos se incluyen como pares nombre/valor, separados por el carácter de igualdad (=) dentro de la etiqueta del elemento. El valor debe delimitarse por comillas simples o dobles.
- ▶ El contenido de un elemento puede ser texto, uno o varios elementos, o la combinación de ambos.

Siguiendo con el mismo ejemplo que en los formatos anteriores, la información de los dos primeros registros podría representarse con el siguiente contenido:

```
&lt;empleados&gt; &lt;empleado&gt; &lt;Nombre&gt;Juan&lt;/Nombre&gt;  
&lt;Edad&gt;45&lt;/Edad&gt; &lt;Cargo&gt;Director&lt;/Cargo&gt;  
&lt;/empleado&gt; &lt;empleado&gt; &lt;Nombre&gt;Antonio&lt;/Nombre&gt;  
&lt;Edad&gt;35&lt;/Edad&gt; &lt;Cargo&gt;Gestor &lt;de  
proyectos&lt;/Cargo&gt; &lt;/empleado&gt; &lt;empleados&gt;
```

Aunque las condiciones para crear los ficheros son claras, estas no siempre garantizan que el fichero XML quede bien formado. Un fichero XML estará bien formado cuando cumpla todas las características de formato, las cuales podrán analizarse mediante alguna herramienta de análisis sintáctico, que incluya la norma correspondiente.

Validación de ficheros XML

Recuerda que todo fichero XML cumple con una jerarquía de etiquetas que garantiza su estructura. Para ello, las etiquetas estarán contenidas adecuadamente unas dentro de otras, con su cierre correspondiente. Recuerda también que el fichero XML debe incluir un único elemento raíz que contendrá las otras etiquetas que existan. El valor de los atributos, por su parte, deben estar entre comillas simples o dobles; y en lo que respecta a las etiquetas, estas son sensibles a mayúsculas y minúsculas.

Cuando un XML cumple con estas consideraciones, se dice que este está «bien

formado». Para que se pueda asegurar que el documento XML es «válido», será necesario validarla mediante algún otro documento que garantice dicha validez.

Validación de ficheros XML mediante DTD

Uno de esos documentos se llama **DTD** (*Document Type Definition* o definición de tipo de documento). Este documento recoge las reglas que debe cumplir el contenido del XML (estructura, elementos y atributos que sí puede incluir el fichero), de ahí que sea útil para garantizar la validez de **todo fichero XML** que haga referencia a él.

Veamos un ejemplo práctico con el siguiente fichero XML:

```
&lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;!DOCTYPE mail  
SYSTEM "ruler.dtd"&gt; &lt;mail&gt; &lt;to&gt;Pepe&lt;/to&gt;  
&lt;from&gt;Boss&lt;/from&gt; &lt;subject&gt;Empty Stock&lt;/subject&gt;  
&lt;body&gt;Remember to buy supplies!&lt;/body&gt; &lt;/mail&gt;
```

Observa cómo en la sentencia DOCTYPE se hace referencia al fichero DTD llamado **ruler.dtd**. Este fichero es el encargado de definir la estructura correcta para el documento llamado «mail» y para las etiquetas que este contiene.

Ejemplo del fichero **ruler.dtd**:

```
&lt;!DOCTYPE mail [ &lt;!ELEMENT mail (to,from,subject,body)&gt;
&lt;!ELEMENT to (#PCDATA)&gt; &lt;!ELEMENT from (#PCDATA)&gt;
&lt;!ELEMENT subject (#PCDATA)&gt; &lt;!ELEMENT body
(#PCDATA)&gt; ]&gt;
```

La forma de interpretar el DTD es el siguiente:

- ▶ : establece que el elemento raíz es mail.
- ▶ : indica los elementos que deben estar contenidos dentro del raíz, esto son to, from, subject y body.
- ▶ : indica el tipo del elemento to, en este caso es #PCDATA.
- ▶ : establece que el elemento from es de tipo #PCDATA.
- ▶ : establece que el elemento subject es de tipo #PCDATA.
- ▶ : establece que el elemento body es de tipo #PCDATA.

Para la DTD, #PCDATA significa de tipo texto.

Aunque los DTD son documentos que se usan cada vez menos, es importante que sepas que estos existen y cuál es su papel de cara a formar y validar ficheros XML.

Validación de ficheros XML mediante esquemas (SCHEMA)

Otra alternativa que existe para validar la estructura y el contenido de los ficheros XML es mediante esquemas. Un esquema, al igual que un DTD, indica la estructura que debe llevar un documento XML. Un fichero esquema lleva la extensión XSD y está definido como un fichero XML, a diferencia de la DTD que utiliza su propio metalenguaje.

Los XSD ofrecen muchas más opciones para declarar la estructura de los ficheros XML. De dichas opciones se puede destacar la extensa lista de tipos de datos predefinidos para elementos y atributos, los cuales a su vez pueden ampliarse o restringirse para crear nuevos tipos.

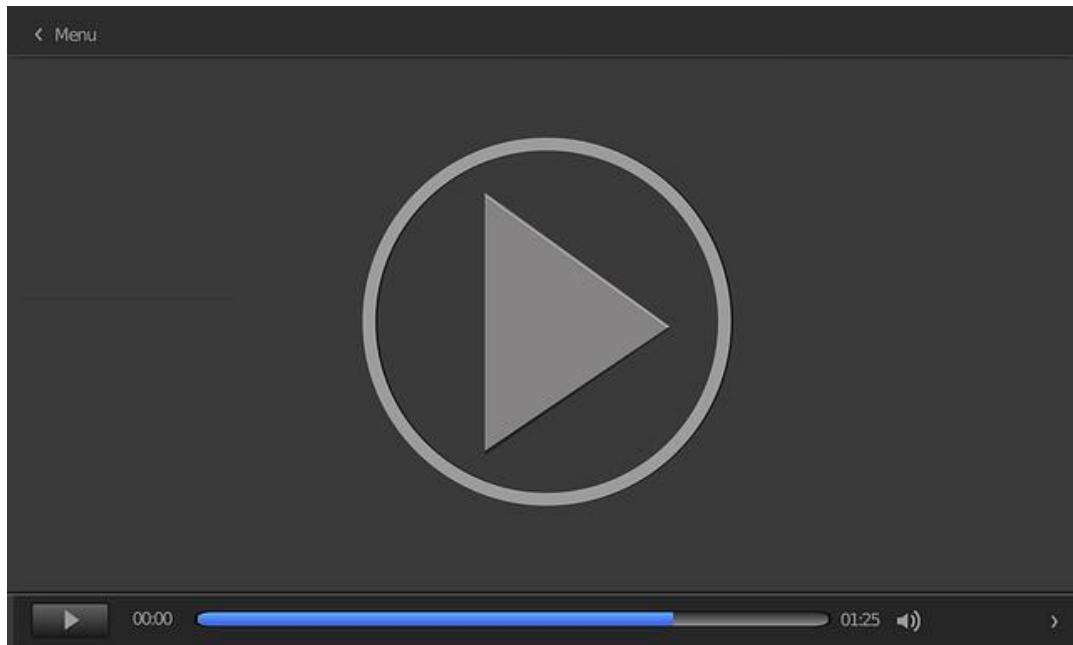
Asimismo, permiten indicar con gran precisión la cardinalidad de un elemento (las veces que dicho elemento puede aparecer en un documento XML). Otra ventaja que tienen es que, gracias a los espacios de nombres, soportan la mezcla de distintos conjuntos de etiquetas (vocabularios).

Siguiendo con el ejemplo anterior, un SCHEMA ideal para nuestro fichero XML de ejemplo podría ser el siguiente:

Ejemplo de fichero **ruler.xsd**:

```
&lt;xs:element name="mail"&gt;          &lt;xs:complexType&gt;
  &lt;xs:sequence&gt;    &lt;xs:element name="to" type="xs:string"/&gt;
    &lt;xs:element name="from" type="xs:string"/&gt;    &lt;xs:element
      name="subject" type="xs:string"/&gt;    &lt;xs:element name="body"
      type="xs:string"/&gt;    &lt;/xs:sequence&gt;    &lt;/xs:complexType&gt;
  &lt;/xs:element&gt;
```

Aprovecha este punto para repasar con la clase «Revisión de formatos CSV, XML y JSON». En ella se presenta un *screencast* con la revisión en detalle de ficheros con los formatos vistos en el tema.



Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=c69be7e8-2c81-499d-ab79-adbe00e8519e>

Vídeo 1. Revisión de formatos CSV, XML y JSON.

Bases de datos

Una **base de datos** es un conjunto de datos persistente utilizado por un sistema de *software*. Siguiendo con las definiciones, y tal y como se menciona en la bibliografía, un sistema de base de datos es un sistema computarizado para el **almacenamiento de registros**. Se pueden mencionar cuatro componentes de un sistema de esta categoría:

- ▶ **Datos.** Los datos en un sistema de base de datos pueden definirse como **integrados**, en aquellos casos en que todos los datos se mantienen unificados y comúnmente serán accedidos por una sola persona, así como **compartidos**, para aquellos casos en los que se desea mantener los conjuntos de datos separados y otorgar privilegios de acceso distintos a varias personas.

- ▶ **Hardware.** Como en otros métodos de almacenamiento, los componentes de hardware que intervienen en un sistema de base de datos son los **volúmenes de almacenamiento**, así como los **procesadores y memoria principal**.
- ▶ **Software.** La capa de *software* entre el usuario y la base de datos física se conoce como DBMS (*Database Management System* o sistema gestor de la base de datos).
- ▶ **Usuarios.** Existen tres clases de usuarios en un sistema de bases de datos:
 - **Programadores:** encargados de crear aplicaciones que permitan la interacción con la base de datos.
 - **Usuarios finales:** utilizan las distintas aplicaciones y herramientas para interactuar con la base de datos.
 - **Administrador de base de datos:** también conocido como DBA por sus siglas en inglés, se encarga de gestionar la estructura, disponibilidad y eficiencia del sistema de base de datos.

En el contexto de bases de datos se utiliza el término *entidad* para describir a cualquier objeto que puede almacenarse en el sistema. Por ejemplo, en una base de datos utilizada por un almacén se puede tener una entidad *producto* para describir los productos disponibles en el almacén y el término *bodega* para describir las bodegas con las que cuenta.

Además, se utilizan los términos *vínculo* o *relación* para representar las relaciones entre las entidades. En el ejemplo del almacén, puede haber una relación bodega-producto para indicar que un producto se almacena en una bodega específica.

Los datos almacenados en una base de datos se pueden categorizar de forma jerárquica. La unidad más pequeña es el campo, el cual suele tener un tipo (número, fecha, etc.) y la base de datos tendrá muchas ocurrencias.

Al conjunto de datos que tienen relación entre sí se le denomina **registro**. Por ejemplo, un registro de tipo producto puede tener campos como «nombre», «precio» y «descripción». Finalmente, al conjunto de registros del mismo tipo se le denomina archivo almacenado.

Bases de datos relacionales y SQL

En un sistema de base de datos **relacional**, los archivos de datos son representados por **tablas**. Cada **columna** de la tabla representa un campo del archivo, mientras que cada **fila** representa un registro de datos. Además, cuando un usuario realiza una operación sobre una tabla, el resultado de dicha operación también será una tabla.

Para exemplificar estos sistemas se ha utilizado el ejemplo de los productos en un almacén. En este caso, la información de los productos puede almacenarse en una tabla **productos** cuyo contenido podría ser el siguiente:

Identificador	Nombre	Precio	Bodega
1	Mesa	150,00	1
2	Silla	50,00	1

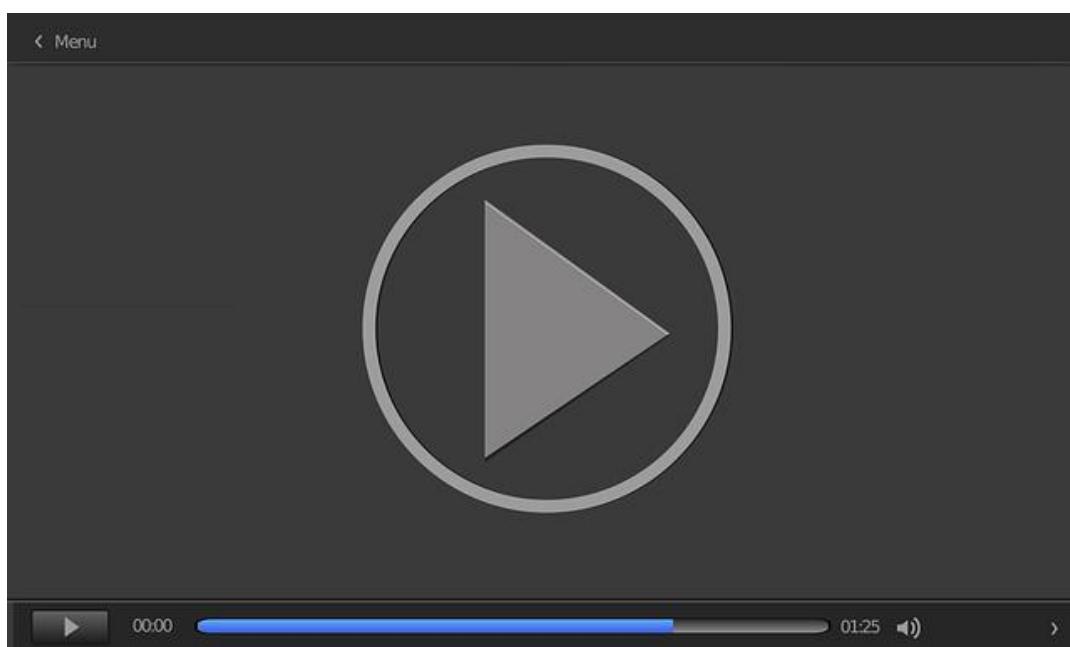
Para interactuar con una base de datos de manera programática, se utiliza el estándar **SQL**. Los comandos para la manipulación de datos se pueden resumir en cuatro:

- ▶ **SELECT**: utilizado para obtener un conjunto de datos a partir de una o varias tablas en un DBMS relacional.
- ▶ **INSERT**: comando para agregar un conjunto de registros dentro de una tabla.
- ▶ **UPDATE**: permite la modificación de un conjunto de campos sobre un conjunto de registros en una tabla específica.

- ▶ **DELETE:** como su nombre indica, permite eliminar un conjunto de registros de una tabla.

En general, las funciones básicas que se pueden llevar a cabo sobre una base de datos se conocen con el acrónimo **CRUD**, que significa *Create, Read, Update and Delete*.

Después de leer gran parte del tema, el reto siguiente es que repases los aspectos más relevantes de los métodos de captura de información con la clase «Resumen de los métodos de captura de información». También será útil recordar las principales características de las bases de datos relacionales y NoSQL.



Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=d8043bb3-2bc5-47fa-93be-aca700e29a9e>

Vídeo 2. Resumen de los métodos de captura de información.

1.4. Casos de estudio

Análisis de *logs* de un servidor web

En este caso de uso se cuenta con un sitio web en el que un equipo de trabajo puede descargar varios documentos. Los documentos se descargan directamente, los usuarios se autentican en el servidor y no existe ninguna aplicación web intermediaria que pueda mantener un registro de los documentos descargados por cada persona.

Se plantea el problema de generar un informe sobre a qué documentos se accede con mayor frecuencia y qué usuarios han tenido mayor actividad con estos documentos. Al no contar con ningún almacenamiento de datos con esta información, se plantea la solución de capturar los datos necesarios mediante el **procesamiento de *logs*** en el servidor web.

Un ejemplo del contenido de los *logs* en un servidor web, en este caso el Servidor HTTP Apache, puede ser el siguiente:

```
192.168.1.101 - juan [21/Nov/2013:12:08:03 +0100] "GET /resource/152  
HTTP/1.1" 200 1368023 "-" "Mozilla/5.0 (X11; Linux x86_64)  
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107  
Safari/537.36" 192.168.1.101 - juan [21/Nov/2013:12:10:22 +0100] "GET  
/resource/601 HTTP/1.1" 200 92349 "-" "Mozilla/5.0 (X11; Linux x86_64)  
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107  
Safari/537.36" 192.168.1.101 - juan [21/Nov/2013:16:36:19 +0100] "GET  
/resource/34 HTTP/1.1" 200 5928327 "-" "Mozilla/5.0 (X11; Linux x86_64)  
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107  
Safari/537.36"
```

Puede observarse que el fichero almacena varios datos sobre la conexión HTTP realizada hacia el servidor: dirección IP, usuario de autenticación, fecha y hora, recurso descargado y el agente o navegador utilizado para la conexión. Todos estos datos pueden capturarse mediante el procesado del fichero y ser almacenados en una base de datos relacional para realizar el análisis correspondiente y generar el informe solicitado.

Web scraping de listado de cursos *online*

El requisito presentado en este caso es capturar la información relacionada a un conjunto de cursos masivos en línea (MOOCs) disponible en el sitio web [Class Central](#). La información de estos cursos se presenta en una serie de tablas.

ADD	COURSE NAME	▲ START DATE	◆ PROVIDER
	6.00x: Introduction to Computer Science and Programming <i>Massachusetts Institute of Technology</i>	26th Sep, 2012	EdX
	6.00x: Introduction to Computer Science and Programming <i>Massachusetts Institute of Technology</i>	4th Feb, 2013	EdX
	6.00x: Introduction to Computer Science and Programming <i>Massachusetts Institute of Technology</i>	16th Oct, 2013	EdX
	Algorithms, Part I <i>Princeton University</i>	12th Aug, 2012	Coursera

Figura 2. Listado de cursos MOOCs WebScraping.

Al no contar con un API que brinde acceso a los datos, la solución ha sido utilizar la herramienta de *web scraping* **Scraper**. Esta herramienta es una extensión del navegador Google Chrome y permite capturar los datos presentados de forma estructurada en formato HTML. Después de capturados, los datos se pueden exportar a una hoja de cálculo del servicio Google Docs.

Acceso a transacciones bancarias mediante API

La tarea solicitada en este caso es crear un conjunto de visualizaciones sobre los patrones de compras de un conjunto de personas. Para esto, se dispone de acceso a información agregada sobre las compras realizadas con tarjeta de débito o crédito en una ciudad.

Con el fin de evitar dar información personal de los clientes del banco, los datos publicados se agrupan por código postal, tipo de establecimiento (alimentación, supermercados, ocio, etc.), día de la semana y hora del día. Los datos disponibles son:

- ▶ Número total de compras.
- ▶ Suma total de las transacciones realizadas dentro de un código postal y por tipo de establecimiento.
- ▶ Diez códigos postales asociados a la tarjeta utilizada y que presentan la mayor frecuencia de aparición.

Los datos se publican a través de un API con servicios web, por lo que el método utilizado se cae en la categoría **acceso a datos públicos**. El primer paso necesario para capturar estos datos ha sido crear una cuenta en el portal de acceso que contiene el catálogo de datos. A continuación, se ha desarrollado una serie de *scripts* para la captura de datos, siguiendo la documentación proporcionada por el mismo portal.

Productos en formato CSV

En este caso, se cuenta con información de un inventario de productos. Los datos capturados de cada producto son el **identificador** (de tipo numérico), el **nombre** (de tipo cadena de texto) y la **cantidad** (de tipo numérico).

Se plantea el problema de definir el formato de un fichero CSV que almacene los datos del inventario. Una de las condiciones presentadas es incluir el nombre de los campos para que no se genere ninguna confusión al procesar el fichero.

Un ejemplo del formato y el contenido del fichero CSV, siguiendo los requisitos planteados, es el siguiente:

```
identificador, nombre, cantidad
1, Plato, 150
2, Sartén, 100
3, Jarra, 200
4, Vaso, 150
```

Como puede observarse, ninguno de los valores incluidos en el campo «nombre» incluye caracteres especiales tales como las comillas dobles, el cambio de línea o la coma. Por esta razón estos valores no están delimitados por comillas dobles.

Información geolocalizada en formato JSON

En esta situación se cuenta con información de la actividad geolocalizada de una persona. Esta característica de geolocalización implica que entre los datos disponibles se encuentra la ubicación del usuario en términos de longitud y latitud, los cuales se representan por un número decimal. Además de la ubicación, se ha capturado un identificador de usuario y el momento de la captura, en formato de fecha y hora.

El requisito en este caso es especificar el formato de un fichero JSON que incluya esta información. Una posible solución es la siguiente:

```
[{"usuario": 1, "ubicacion": {"longitud": 40, "latitud": -3}, "timestamp": "2012-12-01 13:00"}, {"usuario": 2, "ubicacion": {"longitud": 40.1, "latitud": -3.2}, "timestamp": "2012-12-03 10:30"}]
```

Puede observarse que la ubicación del usuario se ha definido como un objeto

perteneciente al de la actividad del usuario. Esta estructura toma ventaja de la recurrencia brindada por el formato JSON.

Información de clientes en una base de datos relacional

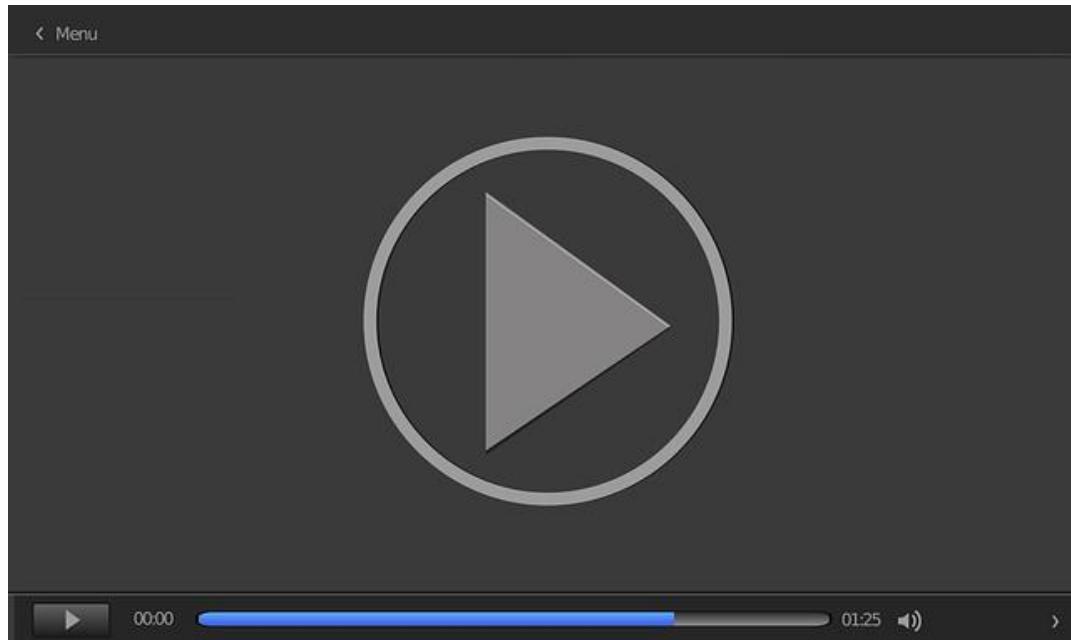
Este último caso contempla el almacenamiento de información sobre clientes en una base de datos relacional. La información para almacenar consiste en un identificador de cliente, el primer nombre, los apellidos y la fecha en la que se suscribieron al servicio.

Al tratarse de una base de datos relacional, la información capturada se representará en la forma de tabla. Así, un ejemplo de esta tabla se puede visualizar en la siguiente estructura.

Ejemplo de datos en tabla			
Identificador	Nombre	Apellidos	Fecha de registro
1	David	Gómez Pérez	2011-01-20
2	Francisco	León García	2011-02-01

Tabla 2. Información de clientes, datos de ejemplo.

En la lección «Captura de información a través de Web Scraping» se presenta un *screencast* de la captura de datos desde una página web utilizando la herramienta Scraper.

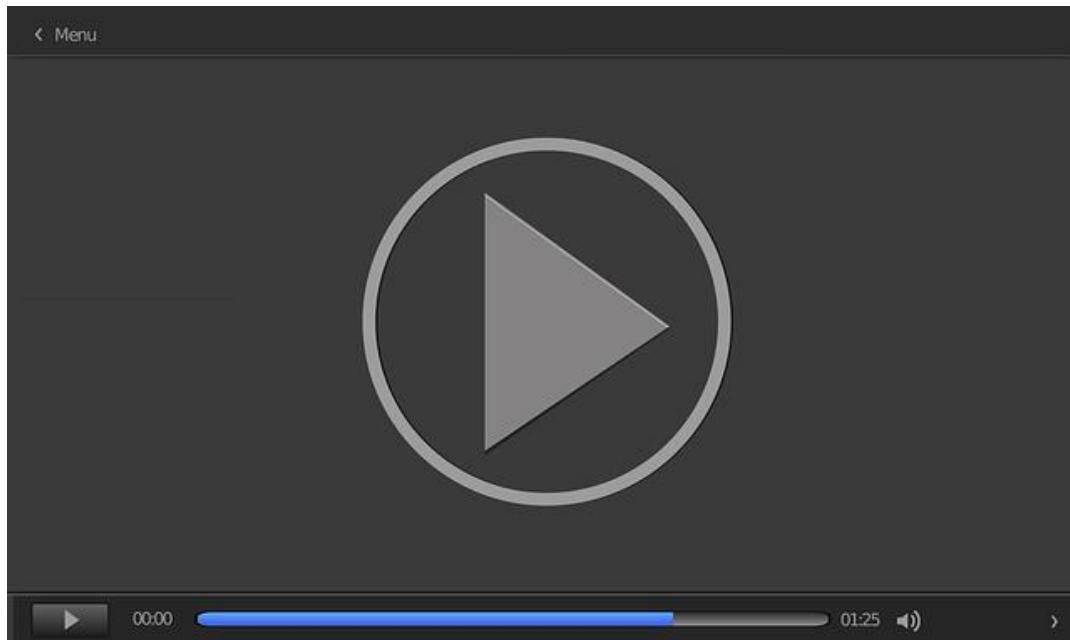


Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=a8af3552-5e97-4a7b-9edf-adbe00e851aa>

Vídeo 3. Captura de información a través de Web Scraping.

En este último vídeo podrás hacer un repaso general de lo visto en este tema, especialmente lo que concierne al origen, la calidad y la organización de los datos.



Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=08017acb-ab88-47a2-b2cb-aca700d73945>

Vídeo 4. Repaso general del primer tema de la asignatura.

1.5. Referencias bibliográficas

Davenport, T., y Prusak, L. (2000). *Working knowledge: how organizations manage what they know*. Harvard Business Review Press.

Jarke, M., Jeusfeld, M. A., Quix, C., y Vassiliadis, P. (1998). Architecture and quality of data warehouses: an extended repository approach. *Advanced Information Systems Engineering, Lecture Notes in Computer Science*, 1413, 243-260.

How do scientists collect data?

Genoino, C. (5 de diciembre de 2012). *How do scientists collect data?* [Archivo de vídeo]. <http://www.youtube.com/watch?v=mPmiW0x3s3k>



Este breve video describe los datos obtenidos por un grupo de científicos especializados en biología marina. Es interesante observar la cantidad de datos contextuales que serán utilizados en el análisis.

Introduction to JSON data

Stanford Dbclass. (26 de diciembre de 2013). *04 01 json intro part1* [Archivo de vídeo]. <https://www.youtube.com/watch?v=lutlvfe8YHU>



JSON Data Introduction



Este vídeo explica detalladamente los aspectos básicos del formato JSON. El vídeo forma parte del curso abierto masivo «Introducción a Base de Datos», ofrecido por la Universidad de Stanford.

The relational model

Stanford Dbclass. (4 de enero de 2013). *02-01-relational-model.mp4* [Archivo de vídeo]. <https://www.youtube.com/watch?v=spQ7IFksP9g>



Otro vídeo del curso «Introducción a Base de Datos» de la profesora Jennifer Widom. En este caso se presenta una explicación muy detallada del modelo de base de datos relacional.

Well-formed XML

Stanford Dbclass. (4 de enero de 2013). *03-01-well-formed-xml.mp4* [Archivo de vídeo]. <https://www.youtube.com/watch?v=x8kMELINaYg>

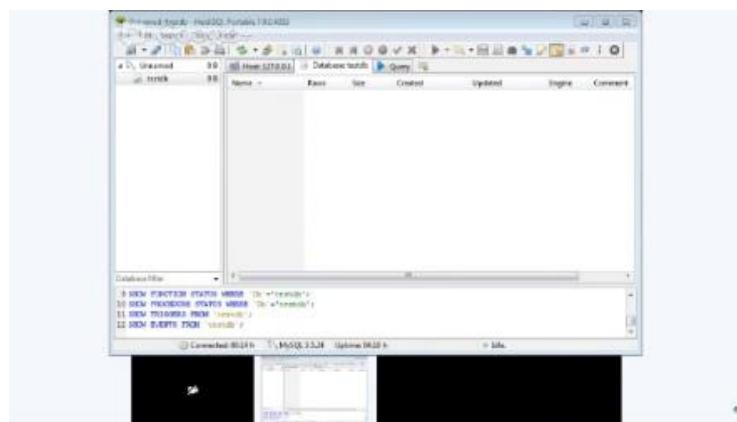


Un último vídeo del curso «Introducción a Bases de Datos» con relación al contenido de este tema. Este vídeo brinda una introducción muy ilustrativa al estándar XML, además de analizar las características de un documento con sintaxis correcta.

Introducción a SQL

Exitae Informática. (25 de abril de 2013). *Introducción a SQL* [Archivo de vídeo].

<http://www.youtube.com/watch?v=fDNgSTF1dVs>



Este vídeo presenta de forma muy guiada el uso de SQL a través de una consola. Se muestran ejemplos tanto de comandos para la creación y modificación de tablas, así como consultas, actualización y eliminación de datos.

Introducción a MySQL

edureka! (23 de octubre de 2018). *MySQL Tutorial For Beginners / Introduction to MySQL* / Learn MySQL / MySQL Training / Edureka [Archivo de vídeo].https://www.youtube.com/watch?v=WmGgxTpGs_8



Este vídeo hace un recorrido por las principales características de uno de los motores de bases de datos SQL gratuitos, muy usado tanto en la industria como en la investigación.

Aunque no es una base de datos que veas en la asignatura, es recomendable que hagas el ejercicio de instalarla e intentar usarla con alguna aplicación para que asimiles su uso de cara a futuros retos profesionales.

7 command-line tools for data science

Janssens, J. (19 de septiembre de 2013). 7 Command-Line Tools for Data Science Workshops [Página web]. <http://jeroenjanssens.com/2013/09/19/seven-command-line-tools-for-data-science.html>

Post en el blog de Jeroen Janssens con una recopilación de herramientas para el tratamiento de ficheros JSON y CSV desde la línea de comando de Unix. Aun si Unix no es la plataforma que utilizas regularmente, es interesante analizar el uso de estos ficheros en un entorno real de analítica de datos (nunca pasan de moda).

SQL for Web Nerds

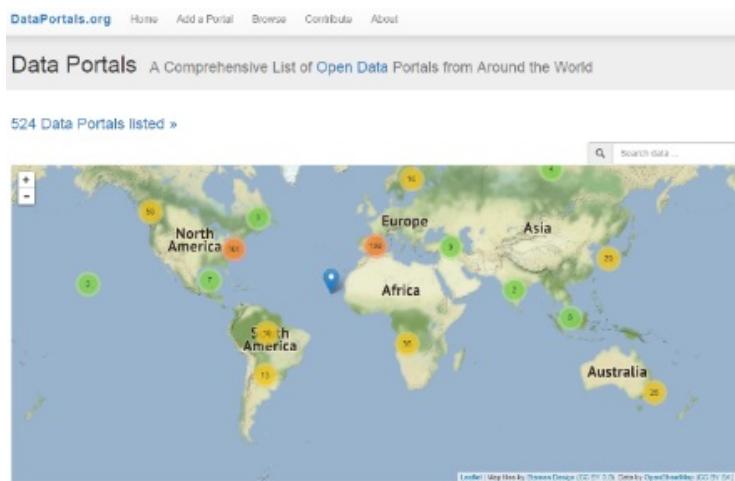
Greenspun, P. (s.f.). SQL for Web Nerdsweb [Página web]. <http://philip.greenspun.com/sql/>

Completo tutorial sobre bases de datos relacionales y SQL. El autor, Philip Greenspun, profesor del Instituto Tecnológico de Massachusetts, describe de forma muy detallada las razones por las que los desarrolladores de aplicaciones web tienen la necesidad de utilizar bases de datos relacionales.

Dataportals.org

Open Knowledge Foundation. (s.f.). Data Portals [Página web]. <http://dataportals.org/>

Listado de catálogos de datos abiertos. Los catálogos están organizados por nivel (local, estatal, nacional, etc.) y por grupos.



Research-Quality Data Sets

Kaggle. (2019). Página web. <https://www.kaggle.com/>

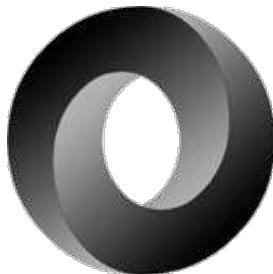
Listado de conjuntos de datos abiertos y con la calidad requerida como para utilizarlos con fines de investigación.



JSON.org

JSON. (s.f.). Introducción a JSON [Página web]. <http://json.org/json-es.html>

Página oficial de la especificación del formato JSON.



Bibliografía

- Chakrabarti, S. (2003). *Mining the web: discovering knowledge from hypertext data* (pp. 17-43). Morgan Kaufmann.
- Debenham, J. (1998). *Knowledge engineering. Unifying knowledge base and database design* (pp. 15-22). Springer.
- Matallah, H., Belalem, G., & Bouamrane, K. (2021). Comparative study between the MySQL relational database and the MongoDB NoSQL database. *International Journal of Software Science and Computational Intelligence (IJSSCI)*, 13(3), 38-63.
- Meier, A., & Kaufmann, M. (2019). *SQL & NoSQL databases*. Springer Fachmedien Wiesbaden.
- de Oliveira, V. F., Pessoa, M. A. D. O., Junqueira, F., & Miyagi, P. E. (2021). SQL and NoSQL Databases in the Context of Industry 4.0. *Machines*, 10(1), 20.
- Redman, T. C. (1996). *Data quality for the information age* (pp. 245-267). Artech House.
- Shafranovich, Y. (2005). Common format and MIME type for Comma-Separated Values (CSV) files. *Internet Engineering Task Force IETF RFC 4180*.
- Sumalatha, A., Vookanti, R., & Vannala, S. (2021). Study on Applications of SQL and Not only SQL Databases used for Big Data Analytics. *International Journal For Research & Development In Technology*, 15, 127-130.
- Wanumen, L. F. (2018). *Bases de datos en SQL server*. Ecoe Ediciones. Disponible en la Biblioteca Virtual de UNIR.

1. ¿Cuál es la unidad semántica mínima que puede almacenarse o comunicarse?

 - A. Dato.
 - B. Información.
 - C. Conocimiento.
 - D. Las respuestas A y B son correctas.

2. ¿Qué métodos pueden utilizarse para la transformación de información a conocimiento?

 - A. Contextualización, agregación y cálculo.
 - B. Repercusión, conexión y conversación.
 - C. Categorización, corrección y agregación.
 - D. Análisis, investigación y discusión.

3. ¿Qué métrica de calidad describe la proporción en la que un conjunto de datos contiene a la población que representa?

 - A. Precisión.
 - B. Consistencia.
 - C. Completitud.
 - D. Interpretabilidad.

4. ¿Cuál de los siguientes es un ejemplo de método de captura manual?

 - A. *Web scraping*.
 - B. Encuestas.
 - C. Acceso a bases de datos relacionales.
 - D. Lectura de termómetro digital.

5. ¿En qué categoría de captura de datos entra la lectura de información del acelerómetro y giroscopio de un teléfono móvil?
 - A. Captura manual.
 - B. Procesamiento de documentos.
 - C. Acceso a datos públicos.
 - D. Sensores.
6. ¿Qué elemento es utilizado para delimitar valores en un fichero CSV?
 - A. Coma.
 - B. CRLF.
 - C. Comillas dobles.
 - D. Espacio.
7. ¿Sobre qué estructuras se basa el formato JSON?
 - A. Objetos y diccionarios.
 - B. Tablas *hash*.
 - C. Objetos y *arrays*.
 - D. Listas enlazadas.
8. ¿Cuál de las siguientes condiciones sobre XML es verdadera?
 - A. Un documento solo puede tener un elemento raíz.
 - B. El contenido de un elemento debe ser otro elemento.
 - C. Todo elemento debe tener un atributo llamado «id».
 - D. Los atributos deben de ser de tipo numérico.

- 9.** ¿Qué nombre recibe un conjunto de datos persistente utilizado por un sistema de *software*?
- A. Archivo.
 - B. Base de datos.
 - C. Registro.
 - D. Las respuestas A y B son correctas.
- 10.** ¿Cuál es la instrucción de SQL para consultar información?
- A. SELECT.
 - B. INSERT.
 - C. UPDATE.
 - D. DELETE.

Bases de Datos para el Big Data

Tema 2. NoSQL

Índice

[Esquema](#)

[Ideas clave](#)

[2.1. Introducción y objetivos](#)

[2.2. Descripción y tipos de bases de datos NoSQL](#)

[2.3. Teorema CAP](#)

[2.4. Elección de base de datos: NoSQL vs. SQL](#)

[2.5. Bases de datos NoSQL](#)

[A fondo](#)

[Building your first MongoDB application](#)

[Entendiendo MongoDB](#)

[MongoDB.com](#)

[Introducción a Apache Cassandra](#)

[Introducción a NoSQL y modelo de datos](#)

[Comparativa NoSQL vs. SQL](#)

[Bibliografía](#)

[Test](#)

NoSQL							
Definición	NoSQL vs. SQL						
El movimiento NoSQL incluye todas las bases de datos con arquitectura distinta a la utilizada en sistemas relacionales tradicionales .	<p>NoSQL:</p> <ul style="list-style-type: none"> - Es una tecnología que se lleva utilizando desde los años 60, aunque el nombre fue acuñado en 2009. - Las bases de datos NoSQL se caracterizan por su escalabilidad horizontal y sencillez, evitan cuellos de botella, permiten manejar grandes volúmenes de datos y su puesta en funcionamiento es más asequible. 						
	<p>Tipos:</p> <ul style="list-style-type: none"> - Las bases de datos NoSQL pueden categorizarse en cuatro grupos: <ul style="list-style-type: none"> - Clave-valor simples, - Clave-valor sofisticadas, - Basadas en documentos, - Basadas en grafos. 						
	<p>Teorema CAP</p> <p>También llamado teorema de Brewer, señala que todo sistema distribuido no puede garantizar a la vez que haya consistencia, disponibilidad y tolerancia a particiones (<i>consistency-availability-partition tolerance</i>).</p>						
	<p>Bases de datos NoSQL</p> <p>Cada base de datos siguiente corresponde a un tipo de NoSQL</p> <table border="1"> <tr> <td>Apache Cassandra</td> <td>Neo4j</td> <td>MongoDB</td> </tr> <tr> <td>Es una base de datos distribuida de código abierto escrita en Java. Todos sus nodos actúan por igual, agrupándose en anillos. Permite sistemas de réplicas y el acceso a los datos se hace a través de CQL.</td><td>Es una base de datos basada en grafos, compatible con ACID. Es accesible desde software escrito en otros lenguajes usando Cypher Query Language, a través de un punto HTTP transaccional.</td><td>Es una base de datos basada en documentos, la cual almacena los datos en esta estructura. El conjunto de documentos se denomina colecciones y el conjunto de estos conforman la base de datos. El formato de almacenamiento es BSON.</td></tr> </table>	Apache Cassandra	Neo4j	MongoDB	Es una base de datos distribuida de código abierto escrita en Java. Todos sus nodos actúan por igual, agrupándose en anillos. Permite sistemas de réplicas y el acceso a los datos se hace a través de CQL.	Es una base de datos basada en grafos, compatible con ACID. Es accesible desde software escrito en otros lenguajes usando Cypher Query Language, a través de un punto HTTP transaccional.	Es una base de datos basada en documentos, la cual almacena los datos en esta estructura. El conjunto de documentos se denomina colecciones y el conjunto de estos conforman la base de datos. El formato de almacenamiento es BSON .
Apache Cassandra	Neo4j	MongoDB					
Es una base de datos distribuida de código abierto escrita en Java. Todos sus nodos actúan por igual, agrupándose en anillos. Permite sistemas de réplicas y el acceso a los datos se hace a través de CQL.	Es una base de datos basada en grafos, compatible con ACID. Es accesible desde software escrito en otros lenguajes usando Cypher Query Language, a través de un punto HTTP transaccional.	Es una base de datos basada en documentos, la cual almacena los datos en esta estructura. El conjunto de documentos se denomina colecciones y el conjunto de estos conforman la base de datos. El formato de almacenamiento es BSON .					

2.1. Introducción y objetivos

Para estudiar este tema se deben leer las Ideas clave que se encuentran a continuación. La mayoría de la literatura relacionada con este sistema se encuentra en inglés, por lo que no es necesario que leas la bibliografía recogida en la sección A fondo, pero sí se sugiere revisarla.

Este tema sirve de introducción a las bases de datos categorizadas como NoSQL. Además de explicar algunos de los conceptos básicos de NoSQL, se introducen algunos de los motores de bases de datos más importantes actualmente. Se recomienda acceder a los enlaces del fabricante si se quiere tener un conocimiento mayor de cada uno de ellos.

Como en la mayoría de las asignaturas relacionadas con herramientas de desarrollo, la mejor manera de estudiar este tema es mediante la realización de ejercicios. La generación de grandes cantidades de datos en varias industrias se ha manifestado en el movimiento y término *big data*. Entre las implicaciones de este movimiento se encuentra la creciente demanda de sistemas de gestión de bases de datos, que permita manejar grandes cantidades de datos.

Hasta hace algunos años, esta demanda se satisfacía mediante el uso de DBMS relacionales en arquitecturas distribuidas, además de un constante análisis de rendimiento. Esta solución se ha quedado corta en algunos casos, por lo que han empezado a surgir nuevas propuestas que se alejan de los típicos sistemas de bases de datos relacionales. En temas posteriores se estudiará con mayor detalle el uso de diferentes bases de datos comúnmente llamadas NoSQL.

Los objetivos que se cubren con en este tema son los siguientes:

- ▶ Comprender los conceptos que describen las bases de datos NoSQL.

- ▶ Diferenciar una base de datos NoSQL y una SQL.
- ▶ Entender el teorema CAP y usarlo como parte del argumento para elegir una base de datos NoSQL como parte de una solución.
- ▶ Diferenciar las bases de datos NoSQL por su tipología.
- ▶ Conocer las principales bases de datos NoSQL del mercado.

2.2. Descripción y tipos de bases de datos NoSQL

El término **NoSQL** se definió en 2009 con el fin de agrupar todas aquellas bases de datos no relacionales que estaban ganando popularidad en ese momento. El término **NoSQL** hace alusión a métodos de almacenamiento no necesariamente estructurados, cuyo lenguaje de consulta «no es SQL» o «no solo es SQL», «**Not only SQL**».

Historia del paradigma NoSQL

Aunque se habla mucho de NoSQL actualmente, no es una tecnología tan nueva como la gente se piensa. El nombre *NoSQL* fue utilizado por Carlo Strozzi en 1998 como nombre de la base de datos basada en archivos que estaba desarrollando. Irónicamente, esta base de datos relacional era solo una interfaz SQL, por lo que no forma parte del movimiento NoSQL actual.

El término volvió a surgir en 2009, cuando Eric Evans lo utilizó para referirse al continuo aumento de bases de datos no relacionales. Aunque el nombre nació en 2009, las bases de datos NoSQL se remontan a la época de las bases de datos de red y jerárquicas y una serie de productos no relacionales que resolvían problemas que nada tienen que ver con los de Amazon, YouTube, Facebook, Twitter, Netflix o Yahoo.

Las bases de datos MultiValue fueron desarrolladas por TRW en 1965. En 1966 se desarrolló en el Hospital Mass General un lenguaje de programación que incorpora una base de datos jerárquica con almacenamiento de árbol B+. En ese mismo año, IBM IMS con Rockwell y Caterpillar desarrollaron una base de datos jerárquica para el programa espacial Apolo.

A lo largo de los años, son muchas las bases de datos que han ido surgiendo, pero si algo ha contribuido al desarrollo de los productos NoSQL, ha sido la serie de *papers*

publicados por Google entre 2003 y 2006 sobre cómo construir una infraestructura escalable para el procesamiento paralelo de grandes volúmenes de datos, que originó Hadoop (y luego Hadoop MapReduce de Yahoo). Más tarde, en 2007 Amazon liberó su servicio de base de datos NoSQL, DynamoDB, con un modelo de datos clave-valor de alta disponibilidad.

En 2012, la cantidad de bases de datos NoSQL ha llegado a ser superior a 120 y cada año están saliendo más y más, con el fin de ocupar su lugar en el mercado o creándose con la necesidad de suplir unas características específicas.

Categorías conocidas

Las bases de datos NoSQL pueden clasificarse en cuatro grandes categorías:

- ▶ **Almacenes de clave-valor simples.** Como su nombre indica, utilizan una clave para acceder a un valor en específico. Los valores almacenados se manejan como *arrays de bytes*, es decir, sin ningún esquema específico asignado. Su aplicación es común en sistemas de caché. Uno de los sistemas más conocidos en esta categoría es **memcached**, el cual es el sistema de facto para la gestión de caché de datos en aplicaciones web.
- ▶ **Almacenes de clave-valor sofisticados.** Estos sistemas son un refinamiento de la categoría anterior con el objetivo de permitir operaciones de lectura y escritura más complejas, así como un modelo de datos ligeramente más elaborado. Ejemplos de sistemas en esta categoría son **Cassandra**, Dynamo, Voldemort y Riak.
- ▶ **Almacenes de documentos.** Los sistemas dentro de esta categoría permiten almacenar estructuras de datos relativamente complejas. Las implementaciones más conocidas en este grupo son CouchDB y **MongoDB**, la cual se estudiará con más detalle.

- ▶ **Almacenes de grafos.** Las bases de datos que entran en esta categoría son las que almacenan la información en estructuras de grafos, cuyos nodos representan la información y las aristas, sus relaciones. Las bases de datos de grafos brindan la facilidad de consultar su información aplicando teoría de grafos.

Dicha teoría comprende un gran número de algoritmos conocidos, que optimizan el recorrido de todos los elementos del grafo (la base de datos), ofreciendo así una navegación mucho más eficiente que las bases de datos relacionales. Algunas implementaciones conocidas son InfoGrid, Virtuoso y **Neo4j**.

Existe otro tipo de bases de datos llamadas **multimodelos**, las cuales combinan características de los cuatro tipos antes descritos. Esta unión de características les permite ofrecer una variedad de funcionalidades muy útiles de cara a la explotación de la información.

El hecho de que soporten varios modelos amplía la forma de interactuar con los datos, independientemente del modelo de datos que se esté utilizando. Esto último brinda a los desarrolladores flexibilidad a la hora de construir sus aplicaciones dentro de la compañía, sin preocuparse de las características del dato.

Dentro de este tipo de bases de datos se encuentra Redis. **Redis** es una base de datos NoSQL multimodelo que permite búsquedas, mensajería, transmisión, grafos y otras capacidades más allá de las que ofrece un simple almacén de datos.

2.3. Teorema CAP

En el apartado anterior se describieron las cuatro grandes categorías que engloban a las bases de datos NoSQL, según la forma de almacenar la información. Existen también otras características que describen aspectos claves del motor de base de datos, tales como que sea escalable (que pueda crecer sin mayores contratiempos) y que además trabaje en entornos distribuidos.

Las bases de datos NoSQL fueron diseñadas precisamente para ser escalables y distribuidas, es por ello por lo que al mencionarlas no se debe pasar por alto un teorema importante que las define según sus capacidades y limitaciones.

Teorema CAP

El teorema **CAP**, o también llamado teorema **Brewer**, indica que todos los sistemas distribuidos no pueden garantizar a la vez que haya consistencia, disponibilidad y tolerancia a particiones (*consistency-availability-partition tolerance*). Estas tres características se definen a continuación:

- ▶ **Consistencia:** sin importar qué servidor reciba la petición, cuando se realiza una consulta o se hace una inserción, el sistema siempre debe devolver la misma información de respuesta.
- ▶ **Disponibilidad:** la caída de uno o más nodos no debe ser un impedimento para que los clientes puedan leer y escribir peticiones.
- ▶ **Tolerancia a particiones:** hace referencia al hecho de que el sistema tiene que seguir funcionando, aunque existan fallos o caídas parciales que dividan el sistema. La división por particiones es una característica de los sistemas distribuidos que permite canalizar grandes flujos de peticiones.

Un ejemplo común de ello son las grandes organizaciones como Netflix, Amazon y Google, entre otras, las cuales crean divisiones geográficas con las que reparten las peticiones de determinadas regiones. Si una región de estas cae, el sistema no debe caer en su totalidad.

Bases de datos NoSQL según el teorema CAP

Las bases de datos NoSQL utilizan diferentes mecanismos para conseguir ser escalables y distribuidas. La variedad de estos mecanismos hace que no todas cumplan los principios del teorema CAP.

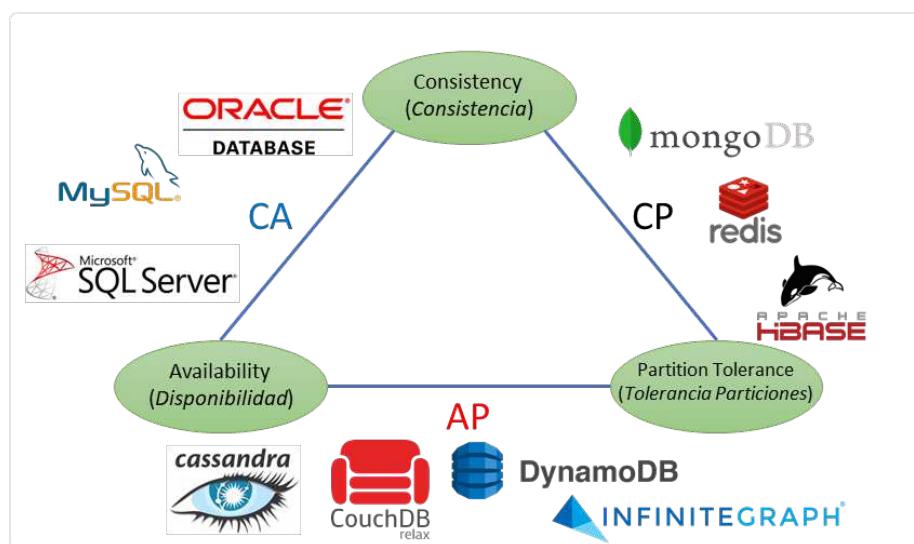


Figura 1. Las bases de datos NoSQL y su relación con el Teorema CAP.

Según sus capacidades, es posible ver cómo las distintas bases de datos cumplen al menos dos de los criterios indicados por el teorema CAP.

- ▶ **CA**: garantizan la consistencia y la disponibilidad en la base de datos, pero no suelen ser óptimas al manejar la tolerancia a particiones. Para cubrir esta deficiencia, lo que suelen hacer es replicar los datos.

- ▶ **AP**: garantizan la disponibilidad y la tolerancia a particiones, pero no suelen ser buenas tratando la consistencia. Determinados motores logran una consistencia parcial mediante técnicas de replicación y verificación.
- ▶ **CP**: garantizan la consistencia y la tolerancia a particiones, pero sacrifican la disponibilidad al replicar los datos entre los distintos nodos con el fin de asegurar la consistencia.

Esta clasificación no siempre será igual para todas las bases de datos NoSQL. Cada cierto tiempo, las actualizaciones de estos productos mejoran las funcionalidades y ofrecen mayores ventajas al usuario con el fin de poder acompañarle en distintas áreas de negocio.

2.4. Elección de base de datos: NoSQL vs. SQL

Uso de NoSQL vs. usos de SQL

Se puede decir que las bases de datos tradicionales son las bases de datos relacionales, que usan un lenguaje estándar para su manipulación y gestión. Su éxito se basa en que es una solución para los problemas de gestión y estructuración de la información de las organizaciones, con un fundamento matemático muy fuerte, lenguaje estandarizado con metodologías estructuradas para el diseño de los sistemas de información y con principios de diseño como la regla **ACID** (Atómica Consistente Aislada y Durable). Estas plataformas tienen muchas herramientas desarrolladas.

Las NoSQL son un conjunto de bases de datos que no se ajustan al modelo de base de datos relacional y se caracterizan por no tener esquema, por no utilizar SQL ni permitir *joins*, por no garantizar la propiedad ACID, por escalar horizontalmente, por hacer uso amplio de la memoria principal del ordenador, por resolver el problema de los altos volúmenes de información y la inmensa cantidad de consultas y transacciones diarias; en resumen, no son relacionales.

Una de las características de NoSQL, como ya se ha dicho antes, es la flexibilidad que proporciona para el uso de modelos de datos, la cual permite tener registros con modelos diferentes. Se puede disponer de registro en el que el atributo «color» aparezca o exista y de registros similares que no contengan dicho atributo, y en otros casos que el atributo sea un *array* o una cadena de caracteres, sin que ocasione ningún tipo de error en base de datos.

El gran problema que tiene esta flexibilidad es la capacidad de introducir errores tanto de desarrollo como de insertado, por lo que el programador tiene una alta responsabilidad al lidiar con los datos, ya que de no hacerlo correctamente dará lugar a alteraciones del modelo que generen errores en la aplicación.

Como punto a favor de esta y otras características, mejoran mucho el rendimiento, principal característica de las arquitecturas NoSQL, pero esto no significa que los datos no respondan a un modelo fijo si así se quiere.

A continuación, se enumeran una serie de ventajas de las bases de datos NoSQL:

- ▶ Responden a la necesidad de escalabilidades horizontal demandada cada vez por más empresas y, además, de manera sencilla.
- ▶ No generan cuellos de botella.
- ▶ Permiten manejar grandes volúmenes de datos.
- ▶ Se pueden tener diferentes bases de datos NoSQL para diferentes proyectos.
- ▶ Equipos económicos para la puesta en marcha.

Por el contrario, tiene una serie de **desventajas** que según la necesidad hacen de su utilización más o menos conveniente:

- ▶ Aunque esté cambiando esta premisa, no ofrecen tanto soporte y nombre como lo hacen bases de datos como Oracle, IBM o Microsoft. Generalmente un vendedor de código abierto no tiene el alcance global, servicios de soporte y la credibilidad de Oracle o IBM.
- ▶ No están lo suficientemente maduras para algunas empresas.
- ▶ Limitaciones de inteligencia de negocios. Por el momento, las bases de datos NoSQL no tienen buena aceptación con las herramientas de BI, lo que origina que una consulta *ad hoc* y su análisis implica conocimientos avanzados de programación. Sin embargo, esta tendencia está cambiando, por ejemplo, *Quest Software* ha creado *Toad* para bases de datos en la nube, que proporciona capacidades de consulta *ad hoc* para algunas bases de datos NoSQL.

- ▶ La falta de experiencia. Debido a que NoSQL es una tecnología novedosa, no hay una gran cantidad de desarrolladores y administradores que la dominen. Esto hace difícil a las empresas encontrar personas con los conocimientos técnicos apropiados.
- ▶ Problemas de compatibilidad. A diferencia de las bases de datos relacionales, que comparten ciertos estándares, las bases de datos NoSQL tienen pocas normas en común. Cada base de datos NoSQL tiene su propia API, las interfaces de consultas son únicas y tienen peculiaridades. Esta falta de normas hace difícil el cambio de unos proveedores a otros.

2.5. Bases de datos NoSQL

Cassandra

Apache Cassandra es una base de datos NoSQL distribuida de código abierto escrita en Java. Está basada en un modelo de almacenamiento clave-valor y se caracteriza en que los nodos que componen el sistema de datos actúan por igual agrupándose en un anillo o clúster. La escalabilidad es lineal y basta con añadir un nuevo nodo al anillo. Permite configurar un sistema de réplicas creando redundancia en sus nodos.

El modelo de datos de Cassandra se compone de:

- ▶ **Esquema:** se encarga del control de acceso y es equivalente a una base de datos.
- ▶ **Tablas:** se compone de un mapa de filas. Una primera tabla define los nombres de las columnas y el tipo de datos que almacenan y las aplicaciones cliente proporcionan las filas siguiendo el esquema de la tabla.
- ▶ **Tipos de datos:** ascii, boolean, blob (contenido de *bytes* arbitrario), composite, counter, timestamp, decimal, double, float, int, variant, bigint, text, cvarchar, uuid.
- ▶ **Filas:** la primera columna una clave de partición.
- ▶ **Columnas:** están compuestas de un *timestamp* y un par clave-valor.

Cassandra Query Language (CQL) es el lenguaje utilizado para proporcionar acceso a datos y es una simplificación del lenguaje SQL. Se puede interactuar con los datos a través de Shell, con herramientas gráficas como devCenter o a través de *drivers* disponibles para la mayoría de los lenguajes de programación.

A continuación, se detallan algunas características que reporta el fabricante:

- ▶ **Probado:** Cassandra es utilizado en empresas como Constant Contact, CERN, Comcast, eBay, GitHub, GoDaddy, Hulu, Instagram, Intuit, Netflix, Reddit, The Weather Channel y más de 1500 empresas más que tienen conjuntos de datos grandes y activos.
- ▶ **Tolerante a fallo:** los datos se replican automáticamente en varios nodos para tolerancia a fallos. Se admite la replicación en varios centros de datos. Los nodos fallidos se pueden reemplazar sin tiempo de inactividad.
- ▶ **Rendimiento:** una de las principales ventajas son sus resultados en *benchmarks* y aplicaciones reales, principalmente debido a las opciones arquitectónicas fundamentales.
- ▶ **Descentralizado:** no hay puntos únicos de fracaso. No hay cuellos de botella en la red. Cada nodo del clúster es idéntico.
- ▶ **Escalable:** algunos de los mayores despliegues en entornos de producción son el de Apple, con más de 75 000 nodos almacenando más de 10 PB de datos, Netflix (2500 nodos, 420 TB, más de 1 billón de solicitudes por día), Easou (270 nodos, 300 TB, más de 800 millones de solicitudes por día) y eBay (más de 100 nodos, 250 TB).
- ▶ **Durable:** Cassandra es adecuado para aplicaciones que no pueden darse el lujo de perder datos, incluso cuando se cae un centro de datos entero.
- ▶ **Control:** se puede elegir entre replicación síncrona o asíncrona para cada actualización. Las operaciones asíncronas altamente asequibles se optimizan con funciones como *Hinted Handoff* y *Read Repair*.
- ▶ **Elástico:** El rendimiento de lectura y escritura aumenta de forma lineal a medida que se agregan nuevas máquinas, sin interrupciones ni interrupciones en las aplicaciones.

- ▶ **Apoyado profesionalmente:** los contratos y servicios de soporte de Cassandra están disponibles a través de terceros.

Neo4j

Es un sistema de gestión de base de datos basada en **grafos**, desarrollado por Neo Technology, Inc en Java lanzada en febrero del 2010. Describo por sus desarrolladores como una base de datos transaccional compatible con ACID con almacenamiento y procesamiento de grafos nativos.

Un grafo se compone de dos elementos: un nodo y una relación. Cada nodo representa una entidad (una persona, un lugar, una cosa, una categoría u otra pieza de datos) y cada relación representa cómo se asocian dos nodos. Esta estructura de propósito general permite modelar todo tipo de escenarios, desde un sistema de carreteras hasta una red de dispositivos, la historia médica de una población o cualquier otra cosa definida por las relaciones.

Una base de datos de grafos es un sistema de gestión de bases de datos en línea con operaciones **Create, Read, Update y Delete (CRUD)** que trabajan en un modelo de datos de grafos. A diferencia de otras bases de datos, las relaciones tienen prioridad en las bases de datos de grafos. Esto significa que su aplicación no tiene que inferir conexiones de datos usando cosas como claves externas o procesamiento fuera de banda, como MapReduce.

El modelo de datos para una base de datos de grafos también es significativamente más simple y expresivo que los de bases de datos relacionales u otras bases de datos NoSQL. Las bases de datos de grafos están diseñadas para utilizarse con sistemas transaccionales (OLTP) y están diseñadas teniendo en cuenta la integridad transaccional y la disponibilidad operativa.

Dos de las propiedades más importantes de esta tecnología son:

- ▶ **Almacenamiento de grafos:** algunas bases de datos de grafos utilizan almacenamiento nativo diseñado específicamente para almacenar y administrar grafos, mientras que otros utilizan bases de datos relacionales u orientados a objetos en su lugar. El almacenamiento no nativo suele ser mucho más latente.
- ▶ **Motor de procesamiento de grafos:** el procesamiento del grafo nativo (también conocido como «adyacencia libre de índice») es el medio más eficiente de procesar los datos de los grafos, ya que los nodos conectados se «apuntan» físicamente entre sí en la base de datos. El procesamiento de grafos no nativos usa otros medios para procesar operaciones CRUD.

Una base de datos de grafos está diseñada específicamente para manejar datos altamente conectados y el aumento en el volumen y la conexión de los datos actuales presenta una tremenda oportunidad para una ventaja competitiva sostenida. Algunas de las principales ventajas de esta base de datos son las siguientes:

- ▶ **Actuación:** para el manejo intensivo de la relación de datos, las bases de datos de grafos mejoran el rendimiento en varios órdenes de magnitud. Con las bases de datos tradicionales, las consultas relacionadas aumentan el tiempo de búsqueda según aumenta el número y la profundidad de las relaciones. Por el contrario, el rendimiento de la base de datos de grafos permanece constante, a medida que sus datos crecen año tras año.
- ▶ **Flexibilidad:** con bases de datos de grafos, los equipos de TI y de arquitectos de datos se mueven a la velocidad de los negocios, porque la estructura y el esquema de un modelo de grafos se amolda a medida que cambian las aplicaciones y las industrias. En lugar de tener que modelar un dominio con antelación, los equipos de datos se pueden agregar a la estructura de grafos existente, sin poner en peligro la funcionalidad actual.

- ▶ **Agilidad:** el desarrollo con bases de datos de grafos se alinea perfectamente con las prácticas de desarrollo ágiles y probadas de hoy, permitiendo que su base de datos evolucione de acuerdo con el resto de la aplicación y con cualquier cambio en los requisitos del negocio.

Algunas de sus principales aplicaciones son:

- ▶ Detección de fraude.
- ▶ Motores de recomendación en tiempo real.
- ▶ Gestión de datos maestros (MDM).
- ▶ Operaciones de red y de TI.
- ▶ Gestión de identidad y acceso (IAM).

Es utilizada en empresas como Walmart, eBay y el Grupo Adidas. Por *startups* como Cobrain, Zephyr Health y Wanderu e incluso sin fines de lucro como el ICIJ y el Foro Económico Mundial, los estudios de casos con bases de datos de grafos abundan con diversidad y profundidad de uso.

Neo4j está disponible en una «edición comunitaria» de código abierto con licencia de GPL3, con copia de seguridad en línea y extensiones de alta disponibilidad con licencia bajo los términos de la Licencia Pública General de Affero. Neo también licencia Neo4j con estas extensiones bajo términos comerciales de código cerrado.

MongoDB

MongoDB (que proviene de *humongous*) es una de las bases de datos NoSQL orientada a documentos desarrollada bajo el concepto de código abierto. Es una de las bases de datos NoSQL más utilizadas en todo el mundo.

El desarrollo de MongoDB comenzó en octubre de 2007 por la compañía de software 10gen. Ahora MongoDB es una base de datos lista para su uso en producción y con muchas características.

Es una base de datos ágil, que permite a los esquemas cambiar rápidamente cuando las aplicaciones evolucionan, proporcionando siempre la funcionalidad que los desarrolladores esperan de las bases de datos tradicionales, tales como índices secundarios, un lenguaje completo de búsquedas y consistencia estricta.

MongoDB ha sido creado para brindar escalabilidad, rendimiento y gran disponibilidad, escalando de una implantación de servidor único a grandes arquitecturas complejas de centros multidatos, mediante la inclusión de más nodos en una red de servidores. Es muy compatible con aplicaciones web.

MongoDB proporciona alto rendimiento, tanto para lectura como para escritura, potenciando la computación en memoria. La replicación nativa de MongoDB y la tolerancia a fallos automática ofrece gran fiabilidad y flexibilidad operativa.

En MongoDB, los datos se almacenan en forma de un **documento**, el cual es una estructura con pares clave-valor, muy similares a los objetos JSON, con la salvedad de que MongoDB almacena los datos en formato **BSON** (representación binaria de JSON). Un documento posee un atributo especial llamado **_id**, el cual indica el valor que identifica de forma inequívoca al documento.

A su vez, los documentos se almacenan en **collections**, formando así un conjunto de documentos con atributos similares. Podría decirse que una *collection* equivale a una tabla en una base de datos relacional. Así, se puede definir una base de datos en MongoDB como un conjunto de *collections*.

Patrones de diseño en MongoDB

Los patrones de diseño en MongoDB pueden agruparse en dos categorías. La primera incluye aquellos patrones que soportan relaciones entre documentos:

- ▶ **Patrón de relación uno-a-uno con documentos embebidos:** en este patrón se embebe un documento dentro de otro con el cual tiene relación. Por ejemplo, se puede contar con un documento de información de un usuario y embeber un subdocumento con información de la dirección del usuario:

```
{ _id: "1", nombre: "Juan", direccion: { ciudad: "Madrid" } }
```

- ▶ **Patrón de relación uno-a-muchos con documentos embebidos:** en este patrón se embebe a varios documentos dentro de otro con el cual tiene relación. Siguiendo con el ejemplo anterior, es probable que el usuario tenga más de una dirección, por lo que se tiene una relación de uno (usuario) a varios (direcciones).

```
{ _id: "1", nombre: "Juan", direcciones: [ { ciudad: "Madrid" }, { ciudad: "Toledo" } ] }
```

- ▶ **Patrón de relación uno-a-muchos con documentos referidos:** este patrón se suele utilizar para evitar repetición en aquellos casos en que un mismo documento se desee embeber en otros varios. Un ejemplo de su aplicación es el siguiente:

```
{ _id: address1, ciudad: "Madrid" }, { _id: address2 ciudad: "Toledo" }, { _id: "1", nombre: "Juan", direcciones: [address1, address2] }
```

- ▶ **Patrón de relación uno-a-uno con documentos referidos:** igual que el anterior, pero solo a un documento. No es aconsejable su utilización, debido a que es más útil utilizar documentos embebidos.

```
{ _id: address ciudad: "Toledo" }, { _id: "1", nombre: "Juan", direcciones: address }
```

El siguiente grupo de patrones está relacionado con modelos basados en estructura de árbol.

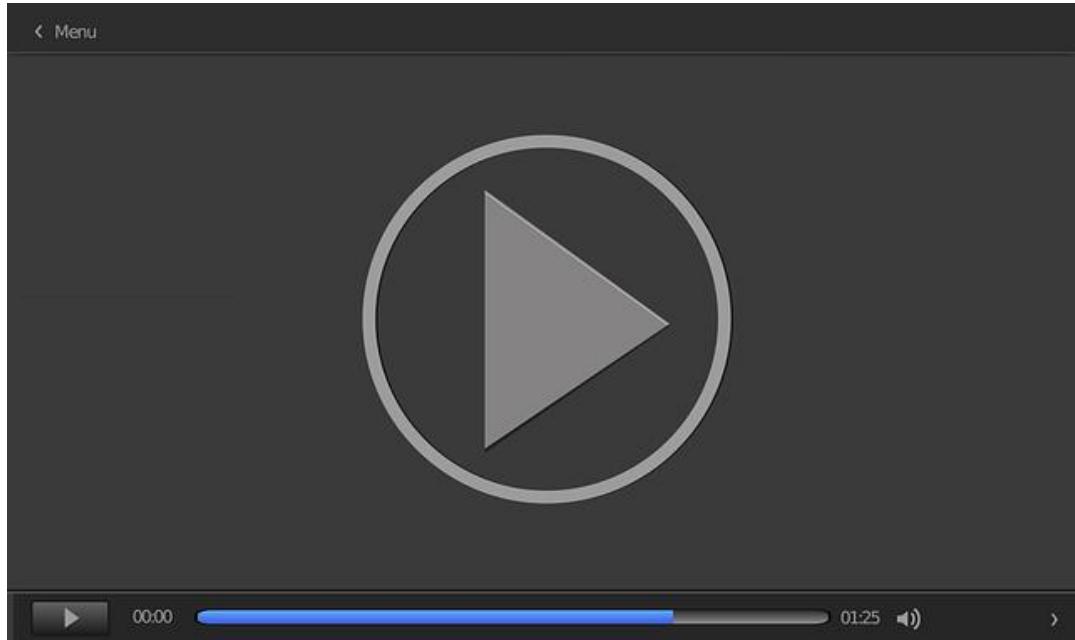
- ▶ **Patrón de modelo de estructura de árbol con referencias al nodo padre .** Tal como indica el nombre, se mantiene un atributo que almacena el _id del nodo padre. A continuación, se muestra un ejemplo con documentos representando carpetas de un sistema de ficheros.

```
{ _id: folder1, nombre: "Raíz", parent: null }, { _id: folder2, nombre: "Folder 2", parent: folder1 } { _id: folder3, nombre: "Folder 3", parent: folder1 }
```

- ▶ **El modelo de estructura de árbol con referencia a nodo hijo.** Como indica el nombre, implica una aproximación inversa al caso de referencias al nodo padre. El ejemplo anterior se vería representado de la siguiente manera.

```
{ _id: folder1, nombre: "Raíz", children: [folder2, folder3] }, { _id: folder2, nombre: "Folder 2", children: [ ] } { _id: folder3, nombre: "Folder 3", children: [ ] }
```

Después de leer todo el tema, el reto siguiente es que repases los aspectos más relevantes de las bases de datos NoSQL con el vídeo «Aspectos relevantes de las bases de datos NoSQL».

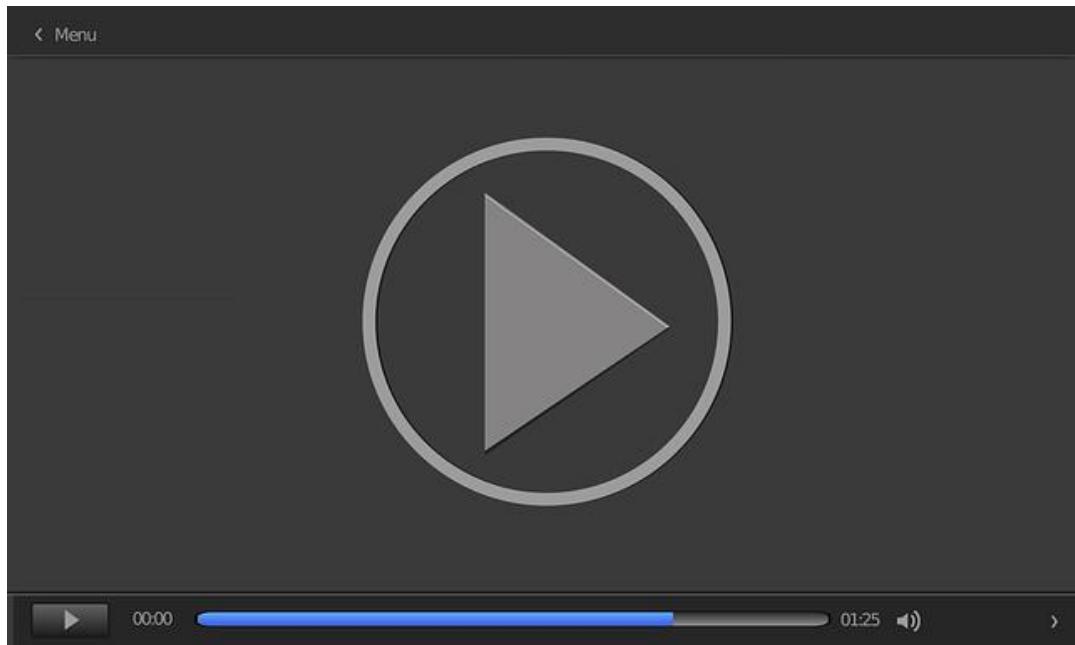


Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=b93f2fb1-390b-496b-9124-aca70108f754>

Vídeo 1. Aspectos relevantes de las bases de datos NoSQL.

En esta otra lección, «Documentos de MongoDB y patrones de diseño», se presenta un *screencast* de la consola de MongoDB, incluyendo ejemplos de documentos y de patrones de diseño.



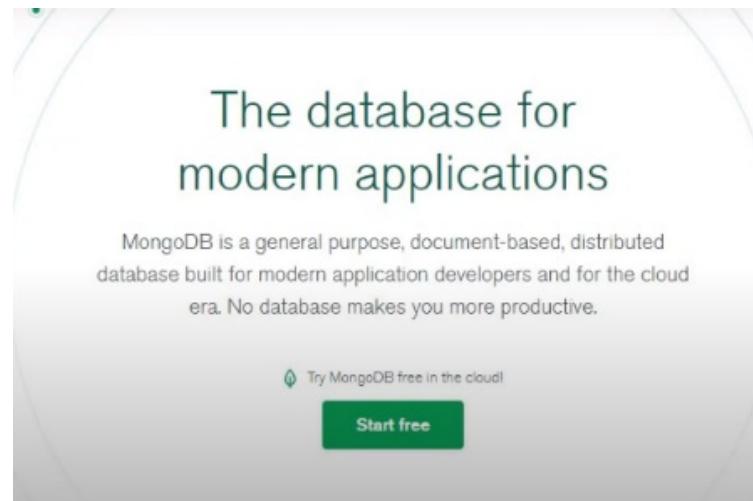
Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=bfe0fc94-a095-4427-9218-adbe00e85197>

Vídeo 2. Documentos de MongoDB y patrones de diseño.

Building your first MongoDB application

Informática DP. (20 de abril de 2020). 1/3 - *MongoDB 2020 - Introducción e Instalación* [Archivo de vídeo]. <https://www.youtube.com/watch?v=w3UhdQ7jffE>



Una serie de tres vídeos con una introducción a MongoDB. En el primero se brinda una introducción a los conceptos principales de MongoDB y sus conceptos equivalentes en una base de datos relacional.

Entendiendo MongoDB

Kane, F. (28 de mayo de 2020). *Understanding MongoDB (2020 Update)* [Archivo de vídeo]. <https://www.youtube.com/watch?v=HWZRio7ukrk>



Durante 17 minutos este vídeo te muestra los aspectos más relevantes de MongoDB, así como las características que presenta MongoDB a nivel de arquitectura. También te cuenta dónde está posicionado MongoDB con respecto al teorema CAP.

MongoDB.com

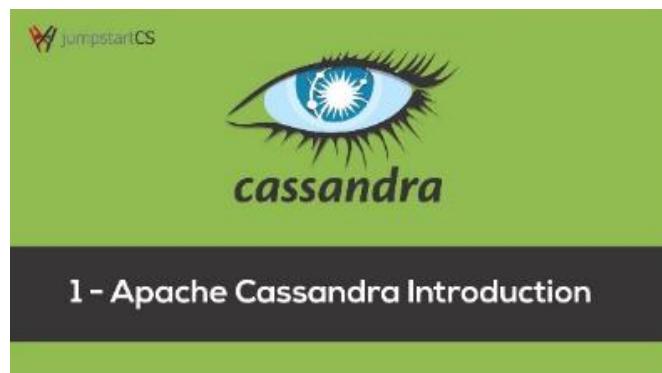
MongoDB. (2021). Documentation [Página web]. <http://docs.mongodb.org/>

Sitio oficial del sistema MongoDB.



Introducción a Apache Cassandra

jumpstartCS. (29 de diciembre de 2019). *Apache Cassandra - Tutorial 1 - Introduction to Apache Cassandra* [Archivo de vídeo]. <https://www.youtube.com/watch?v=s1xc1HVsRk0>



Introducción a la base de datos NoSQL de Apache Cassandra donde se hace una descripción breve de la tecnología y sus características, acompañada con ejemplos básicos de su tecnología de acceso a datos.

Introducción a Neo4j y modelo de datos

Neo4j. (22 de junio de 2020). *Neo4j Introducción y Modelo de Datos* [Archivo de vídeo]. <https://www.youtube.com/watch?v=BcgXw06lSIo>



Introducción a la base de datos NoSQL de Neo4j, donde se hace una descripción amplia de la tecnología y sus características, acompañada con ejemplos básicos del modelado de datos.

Comparativa NoSQL vs. SQL

Be A Better Dev. (10 de febrero de 2020). *SQL vs NoSQL Explained* [Archivo de vídeo]. <https://www.youtube.com/watch?v=ruz-vK8lesE>



Este vídeo oficial de FullStack hace una comparación entre las bases de datos NoSQL y SQL, además de mostrar algunos ejemplos de bases de datos NoSQL.

Bibliografía

- Abed, A. H. (2020). Big Data with Column Oriented NOSQL Database to Overcome the Drawbacks of Relational Databases. *Int. J. Advanced Networking and Applications*, 11(05), 4423-4428.
- Banker, K. (2012). *MongoDB in action* (pp. 3-22, 23-28, 76-126, 241-248). Manning Publications.
- Chodorow, K., y Dirolf, M. (2010). *MongoDB: the definitive guide* (pp. 23-64, 83-92). O'Reilly.
- Contreras Bárcena, D., Gahete Díaz, J. L., Pérez Ferrer, O. L., & Yagüe Juárez, D. (2021). *Sistemas de almacenamiento NoSQL*.
- Copeland, R. (2013). *MongoDB applied design patterns* (pp. 3-15, 37-73). O'Reilly Media.
- Faridoon, A., & Imran, M. (2021). Big Data Storage Tools Using NoSQL Databases and Their Applications in Various Domains: A Systematic Review. *Computing and Informatics*, 40(3), 489-521.
- Meier, A., & Kaufmann, M. (2019). SQL & NoSQL databases. Springer Fachmedien Wiesbaden.
- Moko, A., & Asagba, P. O. (2020). Big Data and NoSQL Databases Architecture: A Review. IIARD—International Institute of Academic Research and Development.
- MUS, M. (2019). Comparison between SQL and NoSQL databases and their relationship with big data analytics.
- Tiwari, S. (2011). *Professional NoSQL* (pp. 3-20, 97-135, 217-232). John Wiley & Sons.

1. ¿Qué tipo de base de datos NoSQL se caracteriza por operaciones de lectura y escritura básicas, además de ser apropiados para entornos de gestión de caché?

 - A. Almacén clave-valor simple.
 - B. Almacén clave-valor sofisticado.
 - C. Base de datos relacional.
 - D. Almacén de documentos.

2. ¿En qué categoría de base de datos NoSQL se clasifica a MongoDB?

 - A. Almacén clave-valor simple.
 - B. Almacén clave-valor sofisticado.
 - C. Base de datos relacional.
 - D. Almacén de documentos.

3. ¿Cuál de las siguientes afirmaciones es correcta?

 - A. Cassandra se caracteriza porque todos sus nodos actúan por igual y se agrupan en anillo.
 - B. Cassandra y Neo4j están desarrolladas en Java.
 - C. Neo4j es una base de datos transaccional compatible con ACID y que almacena y procesa grafos nativos.
 - D. Todas las afirmaciones anteriores son correctas.

4. ¿Cuál es el equivalente a un registro en MongoDB?

 - A. Base de datos.
 - B. *Collection*.
 - C. Tabla.
 - D. Documento.

5. ¿Cuál es el término equivalente a una tabla en MongoDB?
 - A. Base de datos.
 - B. *Collection*.
 - C. Registro.
 - D. Documento.
6. ¿Cuándo se detalla el uso de la primera base de datos NoSQL?
 - A. En 2007, cuando Amazon liberó DynamoDB.
 - B. Con Carlo Strozzi en 1998.
 - C. En 1965 con MultiValue.
 - D. Eric Evans en 2009.
7. ¿Cuál de las siguientes es una ventaja de las bases de datos NoSQL?
 - A. No generan cuellos de botella.
 - B. Tecnología madura.
 - C. Responden a la necesidad de escalabilidades horizontal demandada cada vez por más empresas y, además, de manera sencilla.
 - D. Las respuestas A y C son correctas.
8. ¿Cuál de las siguientes afirmaciones es correcta?
 - A. Todo sistema distribuido no puede garantizar a la vez que haya consistencia, disponibilidad y tolerancia a particiones.
 - B. Un sistema distribuido garantiza al menos disponibilidad y consistencia.
 - C. Un sistema distribuido que garantiza la consistencia y la tolerancia a particiones no sacrifica por ello la disponibilidad.
 - D. Todas las afirmaciones anteriores son correctas

9. ¿Qué patrón de diseño de MongoDB permite incluir un documento dentro de otro?

- A. Uno-a-uno con documentos embebidos.
- B. Uno-a-uno con documentos referidos.
- C. Uno-a-varios con documentos referidos.
- D. Las respuestas B y C son correctas.

10. ¿Qué patrón de diseño de MongoDB permite incluir una lista de referencias a otros documentos dentro de un documento principal?

- A. Uno-a-uno con documentos embebidos.
- B. Uno-a-uno con documentos referidos.
- C. Uno-a-varios con documentos referidos.
- D. Las respuestas B y C son correctas.

Bases de Datos para el Big Data

Tema 3. MongoDB

Índice

[Esquema](#)

[Ideas clave](#)

- [3.1. Introducción y objetivos](#)
- [3.2. Descarga e instalación](#)
- [3.3. Software de apoyo](#)
- [3.4. Flexibilidad del modelo de datos](#)
- [3.5. Inserción de datos](#)
- [3.6. Lectura de datos](#)
- [3.7. Actualización de datos](#)
- [3.8. Caso práctico](#)
- [3.9. Referencias bibliográficas](#)

[A fondo](#)

[MongoDB: the definitive guide](#)

[Bases de datos en MongoDB Compass](#)

[MongoDB: bases de datos, colecciones y documentos](#)

[Documentación oficial de MongoDB](#)

[A cookbook for MongoDB](#)

[MongoBooster](#)

[MongoDB Compass](#)

[Bibliografía](#)

[Test](#)

TRATAMIENTO DE DATOS EN MONGODB				
Software de apoyo	Flexibilidad del modelo	Inserción de datos	Lecturas y consultas	Actualización de datos
Existen muchas aplicaciones, tanto internas como externas, que facilitan el uso del MongoDB.	<p>Elesquema de MongoDB es flexible permitiendo modelos de datos totalmente diferentes.</p> <p>MongoBooster es una herramienta GUI multiplatforma que permite la construcción de consultas fluidas y que posee un gran número de herramientas muy útiles.</p>	<p>El comando <code>db.<collection>.insert</code> recibe el objeto a añadir en formato JSON.</p> <p>El comando <code>db.<collection>.save</code> crea un documento en la collection si este no existía previamente.</p> <p>Dos documentos de una colección pueden tener atributos totalmente diferentes.</p>	<p>El comando principal para la consulta de documentos es <code>db.<collection>.find</code>.</p> <p><code>Find</code> recibe dos parámetros: el criterio de búsqueda y la proyección de atributos a mostrar.</p> <p>En la práctica, la mayoría de documentos de una colección comparten una estructura similar, aunque esto no es una regla fija.</p>	<p>El comando <code>db.<collection>.save</code> actualiza un documento si ya existe en la colección.</p> <p>El comando <code>db.<collection>.update</code> permite cambiar atributos específicos de un conjunto de documentos.</p> <p>El comando <code>update</code> recibe tres parámetros: criterio, acción y opciones.</p>
MongoDB Compass	<p>MongoDB Compass es la solución propietaria de este tipo de herramienta.</p>  <pre>1 { "name": "Big", "path": "public", "path2": "main", "path3": "info" } 2 { "name": "Small", "path": "public", "path2": "main", "path3": "info" } 3 { "name": "Medium", "path": "public", "path2": "main", "path3": "info" }</pre>			
	<p>MongoDB Compass</p>			

3.1. Introducción y objetivos

Este tema se centra en los comandos básicos para manipular datos en MongoDB. La lectura de las Ideas clave te ayudará a conocer la sintaxis de los comandos utilizados para crear, listar y modificar datos.

Al tratarse de un tema muy práctico, la mejor forma de estudiarlo es a través de la realización de los trabajos asignados. Además de los trabajos, es recomendable que trabajes de forma proactiva con MongoDB, lo cual ayudará a familiarizarse con el funcionamiento del sistema, los comandos más relevantes y, especialmente, con los aspectos relacionados con su instalación.

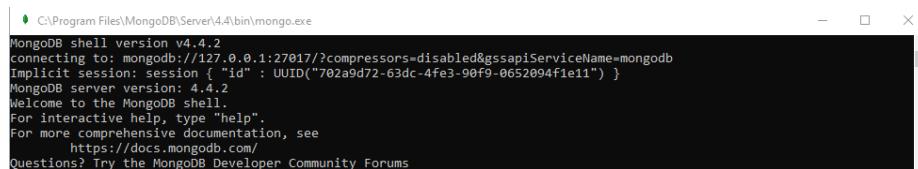
Los objetivos que cubrirás con este tema son los siguientes:

- ▶ Conocer cómo funciona MongoDB dentro del ámbito de las bases de datos NoSQL.
- ▶ Entender la forma en que MongoDB almacena los datos.
- ▶ Aprender las funciones necesarias para la manipulación de los datos.
- ▶ Hacer uso de MongoDB dentro de soluciones de persistencia, aplicando conceptos generales en el modelado de los datos.

Primeros pasos con MongoDB

Para interactuar con el sistema de base de datos MongoDB se utiliza la consola o terminal, la cual se inicia ejecutando el programa Mongo. Es necesario que antes de iniciar la consola se inicie el demonio o servicio *mongod*. El demonio es el encargado de escuchar y servir las peticiones. El lenguaje de programación utilizado en la consola es JavaScript.

Al arrancar la consola, se podrá ver información similar a la siguiente:



```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB shell version v4.4.2
Connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("702a9d72-63dc-4fe3-90f9-0652094f1e11") }
MongoDB server version: 4.4.2
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
    https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
```

Figura 1. Consola de MongoDB.

El mensaje de bienvenida de la consola muestra la versión de la aplicación (en el ejemplo es 4.4.2), seguida de la base de datos a la que se ha conectado. La versión de Mongo que se recomienda para este curso es la 3.6.21.

No hay ningún inconveniente en que instales la versión actual en tu propio ordenador, de esta forma podrás trabajar o bien con la versión Community (más ligera) o bien con la Enterprise (más robusta e ideal para las empresas que requieran de soporte con coste).

En las versiones recientes de MongoDB, además de tener la clásica consola para manipular las bases de datos, estas incorporan una interfaz gráfica web que permite manipular las bases de datos de forma más intuitiva llamada Compass.

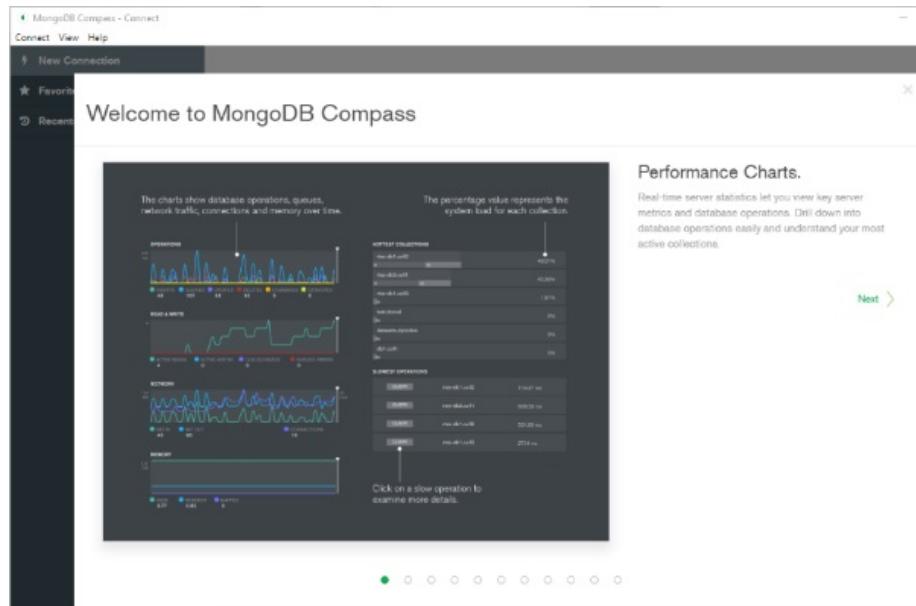


Figura 2. Interfaz MongoDB Compass.

Como se mencionó anteriormente, es muy intuitiva y de forma rápida te podrás hacer al uso de dicha herramienta. Recuerda que Compass necesita conectarse a tu *server* local (o remoto) de MongoDB. El *server* se instalará e iniciará cuando ejecutes el instalador correspondiente.

En el caso de la consola, por defecto esta se conecta a la base de datos llamada *test*. Para cambiar la base de datos sobre la que trabajamos, utilizaremos el comando `use`. Así, en el siguiente ejemplo indicamos a la consola que deseamos utilizar la base de datos *tienda*.

use tienda

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > use tienda
switched to db tienda
MongoDB Enterprise >
```

Figura 3. Uso de base de datos desde MongoDB.

También es posible indicar un nombre de base de datos como argumento del ejecutable *mongo*. Por ejemplo, para obtener el mismo resultado que en el ejemplo anterior se ejecutaría el comando siguiente desde la línea de comandos del sistema operativo.

```
| mongo tienda |
```

Es conveniente aclarar que no es necesario que la base de datos *tienda* exista para poder utilizarla. MongoDB se encargará de crear la base de datos y las *collections* apropiadas cuando el usuario añada nuevos documentos en la misma. Este detalle va en la línea de naturaleza dinámica de este sistema de bases de datos.

3.2. Descarga e instalación

MongoDB ofrece una versión Enterprise y Community de su base de datos de documentos distribuidos. La versión comunitaria ofrece el modelo de documento flexible junto con consultas *ad hoc*, indexación y agregación en tiempo real.

Como sistema distribuido, ofrece una alta disponibilidad a través de la replicación integrada y la conmutación por error junto con la escalabilidad horizontal con fragmentación nativa.

MongoDB Enterprise Server, además de ofrecer todo lo anterior, incluye más funcionalidades relacionadas con la seguridad y distribución de los datos, ideal para entornos profesionales que demanden un servicio con coste por parte del fabricante.

La instalación de MongoDB puede realizarse en el sistema operativo Windows mediante los siguientes pasos:

- ▶ **Paso 1:** descarga el fichero de instalación, en formato zip o msi, desde la siguiente dirección web: <http://www.mongodb.org/downloads>
- ▶ **Paso 2:** en la pestaña Community Server habrá que descargar el archivo msi de instalación. Si el sistema operativo que va a soportar la instalación es de 64 bits, se recomienda elegir la versión de Windows. Esta versión funciona en casi todas las versiones de Windows de 64 bits.
- ▶ **Paso 3:** en el caso de no tener un sistema operativo de 64 bits o querer una versión anterior, en el enlace All Version Binaries se pueden encontrar todas las versiones de la base de datos.
- ▶ **Paso 4:** instalar la aplicación siguiendo los pasos. Es recomendable utilizar un directorio de instalación conocido, por ejemplo, C:/MongoDB/

- ▶ **Paso 5:** inicia el servidor de MongoDB ejecutando el programa mongod.exe, ubicado en la subcarpeta *bin*.
- ▶ **Ten presente:** a partir de la versión 3.x.x de MongoDB, el instalador sobre Windows crea el servicio y lo incluye en la lista de procesos o servicios que ejecutará el sistema operativo. Esto quiere decir que Mongo ya estará funcionando después de instalarlo, solo tendréis que conectaros con su consola o con Compass, el cual viene disponible también y se inicia una vez acaba la instalación.

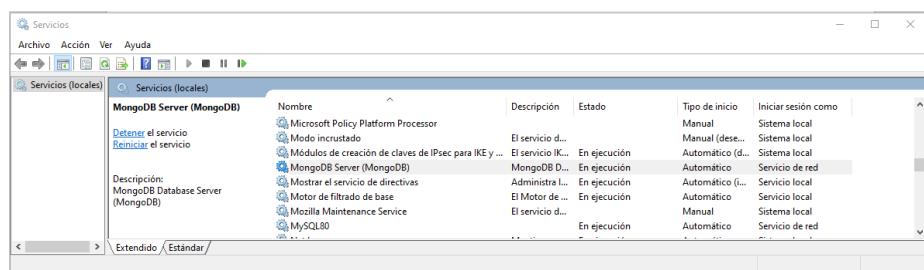


Figura 4. Listado de servicios de Windows, entre ellos el de MongoDB.

Directarios de la instalación

Una vez instalado, en el directorio de instalación podréis encontrar los siguientes subdirectorios:

- ▶ *bin*: todos los programas que necesita Mongo para ejecutarse.
- ▶ *data*: el directorio donde por defecto se crearán las bases de datos.
- ▶ *log*: un fichero con todas las trazas de *log* del motor y de los accesos al servidor mondodb.
- ▶ *snmp*: ficheros de configuración para una instancia de mongod como un subagente SNMP. Esto es útil en la versión Enterprise.

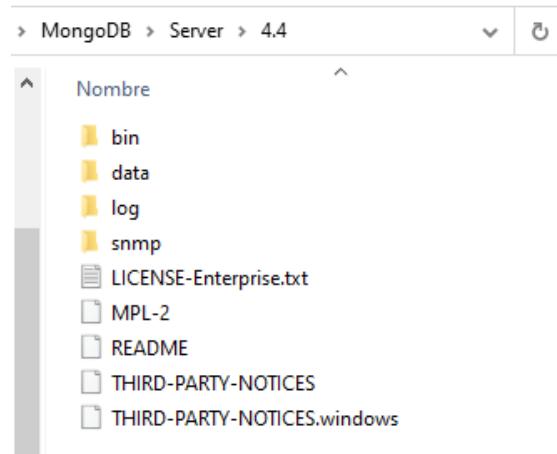


Figura 5. Directorio de instalación de MongoDB.

Algunos programas importantes de la instalación se describen brevemente a continuación.

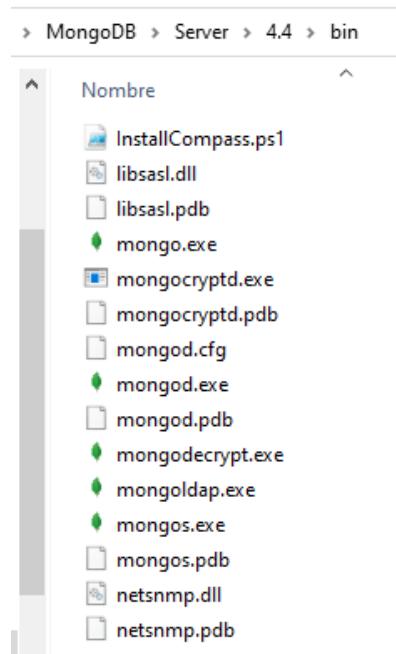


Figura 6. Ejecutables principales de MongoDB.

- ▶ Servidor no como servicio: *mongod.exe*

- ▶ Servidor como servicio: *mongod.exe*, lo utiliza Windows para crear el servicio.
- ▶ Router: *mongos.exe*
- ▶ Cliente: *mongo.exe*
- ▶ Herramientas de monitorización: *mongostat.exe*, *mongotop.exe*
- ▶ Herramienta para importar y exportar bases de datos: *mongodump.exe*,
mongorestore.exe, *mongoexport.exe*, *mongoimport.exe*
- ▶ Otras herramientas: *bsondump.exe*, *mongofiles.exe*

Como recomendación general, la instalación de MongoDB es preferible que se haga sobre un directorio conocido destinado para ello. En Windows se instalará por defecto en Archivos de Programa; esto no es un error, pero es recomendable hacerlo sobre otro directorio menos general.

DB Tools: algunos ejecutables son herramientas de administración de la base de datos que no vienen en la instalación por defecto. Es recomendable descargar dichas herramientas de la web de MongoDB.

Algunos enlaces útiles:

- ▶ [Installation Windows - DB Tools](#)
- ▶ [Download Database-Tools](#)

3.3. Software de apoyo

Para interactuar con MongoDB, no solo se dispone de la consola o terminal clásica (ventana negra) para poder manipular la base de datos. Existe un grupo de aplicaciones con entorno gráfico, tanto propietarias de MongoDB como externas, que facilitan enormemente el uso de esta base de datos, llegando a cubrir muchas de sus funcionalidades.

MongoBooster

MongoBooster es una herramienta GUI multiplataforma basada en *shell* para MongoDB a partir de la versión 2.4, que permite la construcción de consultas fluidas, soporte de sintaxis ES6, una experiencia *IntelliSense* y un gran número de herramientas útiles para el manejo de la base de datos.

El servicio de traducción incorporado conoce todas las posibles complejaciones, métodos, propiedades, variables, palabras clave, incluso los nombres de la colección MongoDB, nombres de campo y operadores. Las sugerencias de *IntelliSense* aparecen mientras se escribe. Siempre se puede activar manualmente con *Ctrl-Shift-Space*.

MongoBooster incluye un lector Shell para hacer MongoDB un poco más fácil. También tiene una utilidad para crear colecciones de forma aleatoria. Posee herramientas de monitorización y estadísticas entre otras.

Schema Analyzer es una herramienta de compilación muy útil. Debido a la característica sin esquema, las colecciones en MongoDB no tienen un documento de esquema para describir el tipo de datos del campo, la estructura de la colección y las validaciones. Con la herramienta Analizador de Esquemas, se puede obtener un documento para describir el esquema de cierto conjunto de registros probados (al azar, el primero, el último, etc.).

MongoBooster tiene varias versiones de pago, pero dispone de una gratuita con ciertas limitaciones:

- ▶ La característica de autocompletar se restringe a la base de datos de «test» o «demo» después de expirar la prueba de 60 días.
- ▶ Schema Analyzer está restringido a la base de datos de test.
- ▶ Deshabilitada la importación de tablas de bases de datos externas.
- ▶ Desactivada la exportación de colecciones a un archivo SQL.
- ▶ Enterprise Auth (X.509, Kerberos, LDAP) se desactiva después de los 60 días de prueba.

MongoDB Compass

MongoDB también posee un *software* propio para la manipulación externa de su base de datos a través de una GUI. Esta aplicación permite ejecutar consultas *ad hoc* en segundos, interactuar con sus datos con funcionalidad CRUD completa y ver y optimizar el rendimiento de la consulta. Está disponible para la mayoría de los sistemas operativos. Compass permite tomar decisiones más inteligentes sobre la indexación, la validación de documentos y mucho más.

MongoDB Compass analiza los documentos y muestra sus estructuras dentro de sus colecciones a través de una GUI intuitiva. Permite visualizar y explorar rápidamente el esquema para comprender la frecuencia, tipos y rangos de campos en su conjunto de datos.

Ofrece estadísticas del servidor en tiempo real, permitiendo conocer las métricas del servidor y las operaciones de la base de datos. Explora las operaciones de la base de datos y ayuda a detectar las colecciones más activas.

Permite construir fácilmente consultas complejas solo pulsando un botón y mostrando los resultados, tanto gráficamente como en documentos JSON. También permite modificar, insertar y borrar documentos a través de un editor visual intuitivo. Se pueden ejecutar las consultas a través de una interfaz gráfica, fácil de entender que ayuda a identificar y resolver problemas de rendimiento. Ayuda a comprender el tipo y el tamaño de los índices, su utilización y propiedades especiales.

Permite crear y modificar las reglas de validación de datos, usando una simple interfaz. El soporte CRUD permite corregir problemas de calidad de datos en documentos individuales.

3.4. Flexibilidad del modelo de datos

MongoDB tiene un **esquema flexible**, donde no es necesario que las colecciones tengan una estructura idéntica para todos los documentos. Esto significa que los documentos de la misma colección no necesitan tener el mismo número de campos o estructura. Cada documento solo necesita contener un número relevante de campos de la entidad u objeto que el documento representa.

En el siguiente ejemplo se pueden ver los datos de 2 pilotos de Fórmula 1. El primero de ellos tiene 2 campeonatos mundiales en esta competición mientras que el segundo ninguno, de ahí que se haya omitido este campo.

```
{  
  "_id" : ObjectId("632c5d85cd35c62d58e36378"),  
  "name" : "Fernando Alonso",  
  "age" : 35,  
  "deportes" : [  
    "Formula 1",  
    "Indi"  
  ],  
  "CampeonatosMundiales" : 2,  
  "hobbies": [  
    "Ciclismo",  
    "Fútbol"  
  ]  
  {  
    "_id" : ObjectId("632c5d85cd35c62d58e11254"),  
    "name" : "Mark Webber",  
    "age" : 40,  
    "deportes" : [  
      "Fórmula 1",  
      "Prototipos"  
    ],  
    "hobbies": [  
      "Cine",  
      "Montañismo"  
    ]  
  }  
}
```

Ideas clave

En la práctica, la mayoría de los documentos de una colección comparte una estructura similar, pero la flexibilidad del esquema aporta una capacidad de modelado de los documentos independientes.

3.5. Inserción de datos

La creación de documentos en una *collection* se realiza con el comando `insert`. La sintaxis del comando es la siguiente:

```
db.<nombre de Collection>.insert(<documento a almacenar>)
```

Por ejemplo, si se desea crear un nuevo documento con información sobre un producto en específico, dentro de la *collection* llamada *productos*, el comando sería el siguiente:

```
db.productos.insert({nombre: "Jabón líquido", precio: 5.50})
```

Este documento creado por este comando tiene dos atributos: nombre y precio. Además, MongoDB crea un atributo especial llamado `_id` de tipo `ObjectId`, el cual es un tipo especial de 12 bytes. `_id` es utilizado para garantizar la unicidad de los documentos en una *collection*.

La siguiente captura muestra la creación de cinco documentos más dentro de la *collection* *productos*.

```
db.productos.insert({nombre: "Barra de Pan", precio: 0.75})
db.productos.insert({nombre: "Zumo de naranja", precio: 1.00})
db.productos.insert({nombre: "Leche", precio: 1.20})
db.productos.insert({nombre: "Queso", precio: 2.00})
db.productos.insert({nombre: "Mantequilla", precio: 1.50})
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.productos.insert({nombre: "Barra de Pan", precio: 0.75})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.productos.insert({nombre: "Zumo de naranja", precio: 1.00})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.productos.insert({nombre: "Leche", precio: 1.20})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.productos.insert({nombre: "Queso", precio: 2.00})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.productos.insert({nombre: "Mantequilla", precio: 1.50})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise >
```

Figura 7. Ejecución de instrucciones de Insert.

Otra alternativa es la inserción múltiple de documentos. Para ello, utilice la instrucción `insertMany([...])` con un *array* de documentos a insertar como argumento.

```
db.productos.insertMany([{nombre: "Barra de Pan", precio: 0.75},
{nombre: "Zumo de naranja", precio: 1.00},
{nombre: "Leche", precio: 1.20},
{nombre: "Queso", precio: 2.00},
{nombre: "Mantequilla", precio: 1.50}])
```

```
MongoDB Enterprise > db.productos.insertMany([{nombre: "Barra de Pan", precio: 0.75},
... {nombre: "Zumo de naranja", precio: 1.00},
... {nombre: "Leche", precio: 1.20},
... {nombre: "Queso", precio: 2.00},
... {nombre: "Mantequilla", precio: 1.50}])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5fcf952d64477ec06ecdbb5d"),
        ObjectId("5fcf952d64477ec06ecdbb5e"),
        ObjectId("5fcf952d64477ec06ecdbb5f"),
        ObjectId("5fcf952d64477ec06ecdbb60"),
        ObjectId("5fcf952d64477ec06ecdbb61")
    ]
}
MongoDB Enterprise >
```

Figura 8. Ejecución de instrucción en MongoDB.

Recuerda que MongoDB utiliza de base JavaScript, por lo tanto, lo siguiente es correcto:

```
try {  
    db.productos.insertMany( [  
        ...  
    ]);  
} catch (e) {  
    print (e);  
}
```

Consulta los documentos creados con el siguiente comando:

```
> db.productos.find()
```



The screenshot shows a terminal window with the following text:

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe  
MongoDB Enterprise > db.productos.find()  
{ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80"), "nombre" : "Barra de Pan", "precio" : 0.75 }  
{ "_id" : ObjectId("5fc9a0d63cb2c79952e59f81"), "nombre" : "Zumo de naranja", "precio" : 1 }  
{ "_id" : ObjectId("5fc9a0dd3cb2c79952e59f82"), "nombre" : "Leche", "precio" : 1.2 }  
{ "_id" : ObjectId("5fc9a0e33cb2c79952e59f83"), "nombre" : "Queso", "precio" : 2 }  
{ "_id" : ObjectId("5fc9a0e93cb2c79952e59f84"), "nombre" : "Mantequilla", "precio" : 1.5 }  
MongoDB Enterprise >
```

Figura 9. Ejecución de instrucción búsqueda en MongoDB.

El `_id` también puede indicarse como parte del documento a crear. En el caso de que el identificador ya exista, el comando `insert` retornará un error, como se muestra en el siguiente ejemplo.

```
> db.productos.insert({ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80"),  
"nombre" : "Barra de Pan", "precio" : 0.75 })
```

Ideas clave

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.productos.insert({ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80"), "nombre" : "Barra de Pan", "precio" : 0.75 })
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 11000,
        "errmsg" : "E11000 duplicate key error collection: tienda.productos index: _id_ dup key: { _id: ObjectId('5fc9a0cc3cb2c79952e59f80') }"
    }
})
MongoDB Enterprise >
```

Figura 10. Ejecución de instrucción en MongoDB.

Inserta nuevamente el documento anterior, pero sin incluir el `_id`.

```
> db.productos.insert({"nombre" : "Barra de Pan", "precio" : 0.75 })
```

```
MongoDB Enterprise > db.productos.insert({"nombre" : "Barra de Pan", "precio" : 0.75 })
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise >
```

Figura 11. Ejecución de instrucción de insert en MongoDB.

¿Qué ocurre?

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.productos.find()
{ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80"), "nombre" : "Barra de Pan", "precio" : 0.75 }
{ "_id" : ObjectId("5fc9a0d63cb2c79952e59f81"), "nombre" : "Zumo de naranja", "precio" : 1 }
{ "_id" : ObjectId("5fc9a0dd3cb2c79952e59f82"), "nombre" : "Leche", "precio" : 1.2 }
{ "_id" : ObjectId("5fc9a0e33cb2c79952e59f83"), "nombre" : "Queso", "precio" : 2 }
{ "_id" : ObjectId("5fc9a0e93cb2c79952e59f84"), "nombre" : "Mantequilla", "precio" : 1.5 }
{ "_id" : ObjectId("5fc9a2603cb2c79952e59f85"), "nombre" : "Barra de Pan", "precio" : 0.75 }
MongoDB Enterprise >
```

Figura 12. Ejecución de instrucción de búsqueda en MongoDB.

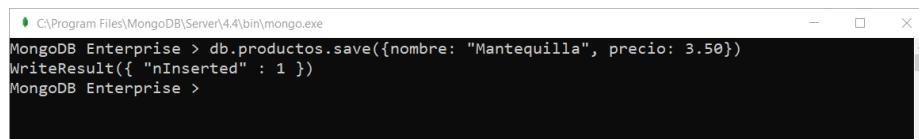
Otro comando para crear documentos es el comando `save`, con la siguiente sintaxis:

```
db.<nombre de Collection>.save(<documento a almacenar>)
```

El comando `save` tiene un comportamiento similar al del comando `insert`. La principal diferencia es que si el documento enviado como argumento contiene un atributo `_id` y existe un documento con dicho `_id`, ese documento será reemplazado

por el nuevo. En la siguiente captura, el documento previo se reemplaza completamente por el nuevo contenido.

```
> db.productos.save({nombre: "Mantequilla", precio: 3.50})
```

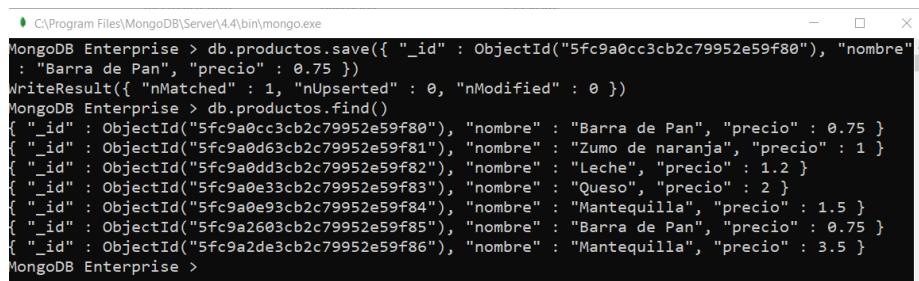


The screenshot shows a terminal window titled 'C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe'. The command entered is 'db.productos.save({nombre: "Mantequilla", precio: 3.50})'. The response from the database is 'WriteResult({ "nInserted" : 1 })'.

Figura 13. Ejecución de instrucción save en MongoDB.

Realiza nuevamente la prueba de insertar el documento indicando el `_id` y observa el **error, ¿o no hay error?**

```
> db.productos.save({ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80") ,  
"nombre" : "Barra de Pan", "precio" : 0.75 })
```



The screenshot shows a terminal window titled 'C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe'. The command entered is 'db.productos.save({ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80") , "nombre" : "Barra de Pan", "precio" : 0.75 })'. The response from the database is 'WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })'.

Figura 14. Ejecución de instrucción de búsqueda en MongoDB.

Usando `save`, cambia el precio del documento anterior y repite el proceso.

```
> db.productos.save({ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80") ,  
"nombre" : "Barra de Pan", "precio" : 1.75 })
```

```
MongoDB Enterprise > db.productos.save({ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80"), "nombre" : "Barra de Pan", "precio" : 1.75 })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
MongoDB Enterprise >
```

Figura 15. Ejecución de instrucción save en MongoDB.

¿Qué ha ocurrido ahora? Sería interesante que lo comentes en el foro.

Ten en cuenta que, a partir de MongoDB 4.2, el método `db.collection.save()` está en desuso, en versiones inferiores sí podrás seguir utilizándolo.

3.6. Lectura de datos

Las consultas son los mecanismos utilizados en bases de datos para leer u obtener datos. En el caso de MongoDB, las consultas solamente pueden obtener datos de una sola *collection*.

El comando utilizado en MongoDB para realizar consultas es `find`, con la siguiente sintaxis:

```
db.<nombre de Collection>.find(<criterios de búsqueda>, <proyección>)
```

Ambos argumentos son opcionales y, en caso de definirlos, deben seguir el formato JSON. Los criterios de búsqueda son las condiciones, que los documentos resultantes deben cumplir. Las condiciones se estructuran en pares clave-valor, donde el valor puede ser de tipo primitivo (número, cadena de texto, etc.) o un objeto para el uso de tres operadores de comparación:

- ▶ **\$gt, \$gte, \$lt, \$lte, \$ne**: para la evaluación de condiciones «mayor que», «mayor o igual que», «menor que», «menor o igual que» y «es diferente a».
- ▶ **\$in, \$nin**: evalúan la pertenencia y ausencia respectivamente de un atributo en un *array* dado.
- ▶ **\$exists**: evalúa que exista el atributo indicado.
- ▶ **\$type**: evalúa que el campo sea de un tipo específico.
- ▶ **\$all**: evalúa *arrays* que coincidan completamente con el parámetro dado.
- ▶ **\$elemMatch**: permite evaluar *arrays* a nivel de elementos, de forma muy específica.

Además, pueden utilizarse los operadores lógicos **\$or, \$and, \$not y \$nor**, los cuales reciben un *array* de expresiones a evaluar en conjunto. MongoDB también brinda

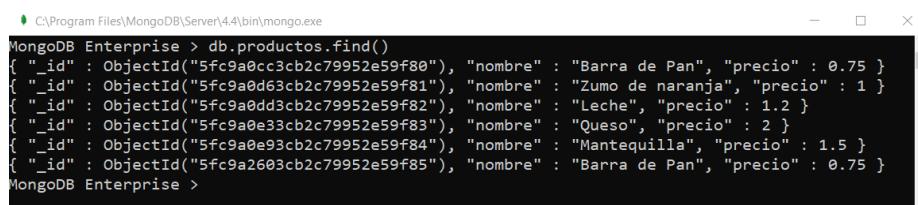
operadores para búsquedas de información geoespacial.

El argumento de proyección indica los atributos. La estructura de este argumento es una lista de pares clave-valor, donde la clave es el atributo y el valor es **1, para mostrarlo, y 0, para ocultarlo**. Por defecto, se muestran todos los atributos de los documentos, y el atributo `_id` se incluye siempre que no es omitido.

En ambos argumentos, se pueden incluir atributos de subdocumentos. Esto se lleva a cabo utilizando el operador punto (.) para separar el subdocumento del atributo.

La siguiente imagen muestra el ejemplo básico del comando `find`, mostrando todos los atributos de los documentos en la *collection* `productos`.

```
> db.productos.find()
```

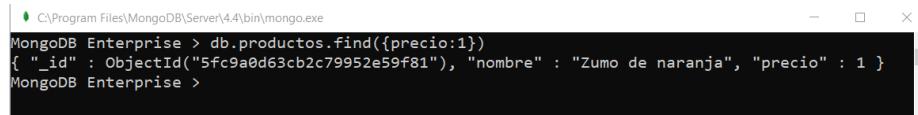


```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.productos.find()
{ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80"), "nombre" : "Barra de Pan", "precio" : 0.75 }
{ "_id" : ObjectId("5fc9a0d63cb2c79952e59f81"), "nombre" : "Zumo de naranja", "precio" : 1 }
{ "_id" : ObjectId("5fc9a0dd3cb2c79952e59f82"), "nombre" : "Leche", "precio" : 1.2 }
{ "_id" : ObjectId("5fc9a0e33cb2c79952e59f83"), "nombre" : "Queso", "precio" : 2 }
{ "_id" : ObjectId("5fc9a0e93cb2c79952e59f84"), "nombre" : "Mantequilla", "precio" : 1.5 }
{ "_id" : ObjectId("5fc9a2603cb2c79952e59f85"), "nombre" : "Barra de Pan", "precio" : 0.75 }
MongoDB Enterprise >
```

Figura 16. Ejecución de instrucción de búsqueda en MongoDB.

El siguiente ejemplo muestra todos los productos con el atributo `precio` exactamente igual a 1.

```
> db.productos.find({precio:1})
```



```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.productos.find({precio:1})
{ "_id" : ObjectId("5fc9a0d63cb2c79952e59f81"), "nombre" : "Zumo de naranja", "precio" : 1 }
MongoDB Enterprise >
```

Figura 17. Ejecución de instrucción de búsqueda en MongoDB.

Mientras que en el siguiente ejemplo se muestran tres consultas. La primera muestra los productos con precio menor o igual que 1. La segunda muestra los productos que

cumplen la condición de tener precio mayor que 1 y menor que 2. La tercera consulta retorna el mismo resultado que la segunda, pero utilizando el operador \$and.

```
db.productos.find({precio: {$lte: 1} })
db.productos.find({precio: {$gt: 1, $lt: 2} })
db.productos.find({ $and: [{precio: {$gt: 1}}, {precio: {$lt: 2}} ]})
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.productos.find({precio: {$lte: 1} })
{ "_id" : ObjectId("5fc9a0d63cb2c79952e59f81"), "nombre" : "Zumo de naranja", "precio" : 1 }
{ "_id" : ObjectId("5fc9a2603cb2c79952e59f85"), "nombre" : "Barra de Pan", "precio" : 0.75 }
MongoDB Enterprise >
MongoDB Enterprise > db.productos.find({precio: {$gt: 1, $lt: 2} })
{ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80"), "nombre" : "Barra de Pan", "precio" : 1.75 }
{ "_id" : ObjectId("5fc9a0dd3cb2c79952e59f82"), "nombre" : "Leche", "precio" : 1.2 }
{ "_id" : ObjectId("5fc9a0e93cb2c79952e59f84"), "nombre" : "Mantequilla", "precio" : 1.5 }
MongoDB Enterprise >
MongoDB Enterprise > db.productos.find({ $and: [{precio: {$gt: 1}}, {precio: {$lt: 2}} ]})
{ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80"), "nombre" : "Barra de Pan", "precio" : 1.75 }
{ "_id" : ObjectId("5fc9a0dd3cb2c79952e59f82"), "nombre" : "Leche", "precio" : 1.2 }
{ "_id" : ObjectId("5fc9a0e93cb2c79952e59f84"), "nombre" : "Mantequilla", "precio" : 1.5 }
MongoDB Enterprise >
```

Figura 18. Ejecución de instrucción de búsqueda en MongoDB.

Todos los ejemplos previos han mostrado los documentos completos. El siguiente ejemplo muestra solamente el atributo nombre para la última consulta.

```
> db.productos.find({ $and: [{precio: {$gt: 1}}, {precio: {$lt: 2}} ]},
{_id: 0, nombre: 1})
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.productos.find({ $and: [{precio: {$gt: 1}}, {precio: {$lt: 2}} ]}, {_id: 0, nombre: 1})
{ "nombre" : "Barra de Pan" }
{ "nombre" : "Leche" }
{ "nombre" : "Mantequilla" }
MongoDB Enterprise >
```

Figura 19. Ejecución de instrucción de búsqueda con proyección en MongoDB.

Los resultados obtenidos a partir de una consulta pueden transformarse mediante cualquier combinación de tres operaciones:

- ▶ **limit**: indica la cantidad máxima de documentos a mostrar.
- ▶ **skip**: indica la cantidad de documentos a omitir al inicio en un inicio.
- ▶ **sort**: indica, a través de pares clave-valor donde la clave es el atributo sobre el que

se ordenará y el valor es 1 para un orden ascendente y -1 para un orden descendente.

El siguiente ejemplo muestra el uso de `sort` y `limit` para mostrar el producto con mayor precio en la *collection*.

```
> db.productos.find().sort({precio: -1}).limit(2)
```

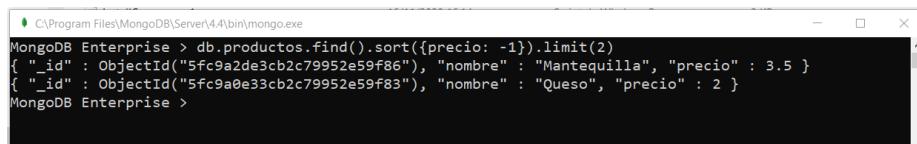
A screenshot of a terminal window titled 'C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe'. The window contains the command 'db.productos.find().sort({precio: -1}).limit(2)' and its output, which shows two documents: one for 'Mantequilla' at 3.5 and another for 'Queso' at 2.

Figura 20. Ejecución de instrucción de búsqueda ordenada en MongoDB.

Además de `find`, MongoDB brinda el comando `findOne` para la búsqueda de un solo documento en una consulta. Los argumentos son los mismos que en el comando `find`

3.7. Actualización de datos

En el subtema sobre inserción de datos se menciona que el comando puede utilizarse para reemplazar un documento específico en una *collection*. Si lo deseado es modificar documentos que coinciden con un criterio de búsqueda, puede utilizarse el comando .

Antes de la versión 2.2:

```
db.<nombre de collection>.update(<criterios de búsqueda>, <documento con cambios>, <valor de upsert>, <valor de multi>)
```

A partir de la versión 2.2:

```
db.<nombre de collection>.update(<criterios de búsqueda>, <documento con cambios>, <opciones de modificación>)
```

Los criterios de búsqueda del comando update siguen las reglas indicadas para el comando `find`. Por defecto, el documento que cumpla con esos criterios se reemplaza por el documento enviado como segundo argumento. Si lo deseado es realizar un cambio específico, se pueden utilizar los siguientes operadores:

- ▶ **\$set**: permite crear o cambiar un atributo en específico.
- ▶ **\$unset**: permite eliminar un atributo.
- ▶ **\$inc**: incrementa un número.
- ▶ **\$rename**: cambia el nombre de un atributo.

Las opciones de modificación pueden ser dos:

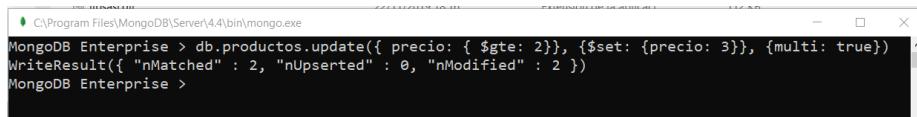
- ▶ **multi**: por defecto, `update` solamente modifica un documento. Si el modificador `multi` tiene el valor `true`, se modificarán todos los documentos que cumplan con las condiciones de búsqueda.

- ▶ **upsert:** si ningún documento cumple con las condiciones de búsqueda y `upsert` se define con el valor `true`, el comando `update` creará un documento.

En cualquier modificación, el atributo `_id` se mantiene constante siempre que no se modifique directamente.

El siguiente ejemplo muestra el uso del comando `update` para cambiar el precio de todos los productos con precio mayor o igual a 2 a un fijo de 3.

```
> db.productos.update({ precio: { $gte: 2}}, { $set: {precio: 3}}, {multi: true})
```



A screenshot of a terminal window titled 'C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe'. The window contains the following text:
MongoDB Enterprise > db.productos.update({ precio: { \$gte: 2}}, { \$set: {precio: 3}}, {multi: true})
WriteResult({ "nMatched": 2, "nUpserted": 0, "nModified": 2 })
MongoDB Enterprise >

Figura 21. Ejecución de instrucción de actualización en MongoDB.

Realiza ahora la siguiente prueba:

```
> db.productos.updateMany({ precio: { $gte: 2}}, { $set: {precio: 7.5}})
```

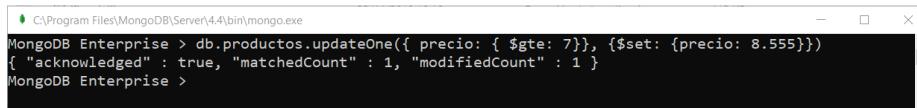


A screenshot of a terminal window titled 'C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe'. The window contains the following text:
MongoDB Enterprise > db.productos.updateMany({ precio: { \$gte: 3}}, { \$set: {precio: 7.5}})
{ "acknowledged": true, "matchedCount": 2, "modifiedCount": 2 }
MongoDB Enterprise > C:\Users\mfcardenas\Desktop

Figura 22. Ejecución de instrucción de actualización masivo en MongoDB.

Y ahora esta otra prueba:

```
> db.productos.updateOne({ precio: { $gte: 7}}, { $set: {precio: 8.555}})
```



A screenshot of a terminal window titled 'C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe'. The window contains the following text:
MongoDB Enterprise > db.productos.updateOne({ precio: { \$gte: 7}}, { \$set: {precio: 8.555}})
{ "acknowledged": true, "matchedCount": 1, "modifiedCount": 1 }
MongoDB Enterprise >

Figura 23. Ejecución de instrucción de actualización única en MongoDB.

```
> db.productos.find()
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.productos.find()
{ "_id" : ObjectId("5fc9a0cc3cb2c79952e59f80"), "nombre" : "Barra de Pan", "precio" : 1.75 }
{ "_id" : ObjectId("5fc9a0d63cb2c79952e59f81"), "nombre" : "Zumo de naranja", "precio" : 1 }
{ "_id" : ObjectId("5fc9a0dd3cb2c79952e59f82"), "nombre" : "Leche", "precio" : 1.2 }
{ "_id" : ObjectId("5fc9a0e33cb2c79952e59f83"), "nombre" : "Queso", "precio" : 8.555 }
{ "_id" : ObjectId("5fc9a0e93cb2c79952e59f84"), "nombre" : "Mantequilla", "precio" : 1.5 }
{ "_id" : ObjectId("5fc9a2603cb2c79952e59f85"), "nombre" : "Barra de Pan", "precio" : 0.75 }
{ "_id" : ObjectId("5fc9a2de3cb2c79952e59f86"), "nombre" : "Mantequilla", "precio" : 7.5 }
MongoDB Enterprise >
```

Figura 24. Ejecución de instrucción de búsqueda en MongoDB.

La eliminación de documentos se realiza a través del comando `remove`, `deleteMany` y `deleteOne`.

El primer argumento se comporta de la misma manera que los criterios de búsqueda en el comando `find`, mientras que el segundo es un valor booleano para indicar si se desea eliminar solamente un documento de la *collection*.

Inserta los siguientes documentos:

```
db.inventario.insertMany([
  {item : "revista" , cantidad : 25 , estado : "A" },
  {item : "libro" , cantidad : 50 , estado : "B" },
  {item : "postal" , cantidad : 45 , estado : "A" }
]);
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.inventario.insertMany([
...   { item : "revista" , cantidad : 25 , estado : "A" },
...   { item : "libro" , cantidad : 50 , estado : "B" },
...   { item : "postal" , cantidad : 45 , estado : "A" }
... ]);
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5fc9acbd3cb2c79952e59f87"),
    ObjectId("5fc9acbd3cb2c79952e59f88"),
    ObjectId("5fc9acbd3cb2c79952e59f89")
  ]
}
MongoDB Enterprise >
```

Figura 25. Ejecución de instrucción de actualización masivo en MongoDB.

Ahora ejecuta la instrucción siguiente y borra todos los documentos cuyo estado sea “B” .

```
db.inventario.deleteMany({ estado : "B" })
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.inventario.deleteMany({ estado : "B" })
{ "acknowledged" : true, "deletedCount" : 1 }
MongoDB Enterprise >
```

Figura 26. Ejecución de instrucción de borrado masivo en MongoDB.

Consulta los documentos y comprueba que solo quedan los documentos con estado igual a “ A ” .

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.inventario.deleteMany({ estado : "B" })
{ "acknowledged" : true, "deletedCount" : 1 }
MongoDB Enterprise > db.inventario.find()
[ { "_id" : ObjectId("5fc9acbd3cb2c79952e59f87") , "item" : "revista" , "cantidad" : 25 , "estado" : "A" },
  { "_id" : ObjectId("5fc9acbd3cb2c79952e59f89") , "item" : "postal" , "cantidad" : 45 , "estado" : "A" } ]
MongoDB Enterprise >
```

Figura 27. Ejecución de instrucción de borrado masivo en MongoDB.

Ahora ejecuta el otro comando de borrado y observa cómo se borra solo un único documento.

```
db.inventario.deleteOne({ estado : "A" })
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.inventario.deleteMany({ estado : "B" })
{ "acknowledged" : true, "deletedCount" : 1 }
MongoDB Enterprise > db.inventario.find()
[ { "_id" : ObjectId("5fc9acbd3cb2c79952e59f87"), "item" : "revista", "cantidad" : 25, "estado" : "A" },
  { "_id" : ObjectId("5fc9acbd3cb2c79952e59f89"), "item" : "postal", "cantidad" : 45, "estado" : "A" } ]
MongoDB Enterprise > db.inventario.deleteOne({ estado : "A" })
{ "acknowledged" : true, "deletedCount" : 1 }
MongoDB Enterprise > db.inventario.find()
[ { "_id" : ObjectId("5fc9acbd3cb2c79952e59f89"), "item" : "postal", "cantidad" : 45, "estado" : "A" } ]
MongoDB Enterprise >
```

Figura 28. Ejecución de instrucción de borrado masivo en MongoDB.

¿Has podido comprender cuándo conviene usar un comando antes que otro?

3.8. Caso práctico

Para este caso, descarga e importa con MongoDB Compass el fichero [JSON con datos de prueba](#). Si el fichero no está disponible, escribe al profesor para que os lo publique en el aula virtual.

Una vez cargado en MongoDB, se realizarán una serie de ejercicios prácticos con los que es posible aprender muchas de las funcionalidades descritas anteriormente. Se va a partir de una colección que almacene datos de libros. Un ejemplo de registro es el que se puede ver a continuación:

```
{  
    "_id": 3,  
    "title": "Specification by Example",  
    "isbn": "1617290084",  
    "pageCount": 0,  
    "publishedDate": { "$date": "2011-06-03T07:00:00.000Z"},  
    "thumbnailUrl": "https://....",  
    "shortDescription": "...",  
    "longDescription": "...",  
    "status": "PUBLISH",  
    "authors": ["Gokko Adzic"],  
    "categories": ["Software Engineering"]  
}
```

Nota: los campos en rojo se irán omitiendo en las consultas debido a su amplio contenido.

Para iniciar las consultas se va a mostrar los 3 primeros registros con un simple comando `find` para buscar y `limit(3)` para restringir el número de búsquedas a 3. Para ello ejecutamos `db.books.find().limit(3)`. El resultado es el que se puede ver a continuación.

```
> db.books.find({}, {thumbnailUrl: 0, longDescription: 0, shortDescription: 0 }).limit(3)
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.books.find({}, {thumbnailUrl: 0, longDescription: 0, sh
{ "_id" : 1, "title" : "Unlocking Android", "isbn" : "1933988673", "pageCount"
, "authors" : [ "W. Frank Ableson", "Charlie Collins", "Robi Sen" ], "categorie
{ "_id" : 2, "title" : "Android in Action, Second Edition", "isbn" : "193518272
tus" : "PUBLISH", "authors" : [ "W. Frank Ableson", "Robi Sen" ], "categories"
{ "_id" : 3, "title" : "Specification by Example", "isbn" : "1617290084", "page
LISH", "authors" : [ "Gojko Adzic" ], "categories" : [ "Software Engineering" ]
MongoDB Enterprise >
```

Figura 29. Ejecución de instrucción de búsqueda con proyección en MongoDB.

Antes de trabajar con los filtros, se van a recuperar todos los autores que tiene disponible la base de datos. Para ello se utiliza el comando `distinct`.

```
> db.books.distinct("authors")
```

Como se puede ver, no se repite ninguno de ellos:

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.books.distinct("authors")
[
    "",
    "Adam Benoit",
    "Adam Chace",
    "Adam Machanic",
    "Adam Tacy",
    "Adele Goldberg",
    "Ahmed Sidky",
    "Ajamu A. Wesley",
    "Ajay Kapur",
    "Alan Dennis",
    "Alan Mycroft",
```

Figura 30. Función `distinct` en MongoDB.

Para realizar búsquedas utilizando el comando `distinct`, hay que aplicar el filtro en el segundo parámetro de la función como se puede ver a continuación:

```
> db.books.distinct("title", {authors: "Vlad Landres"})
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.books.distinct("title", {authors: "Vlad Landres"})
[
    "SQR in PeopleSoft and Other Applications",
    "SQR in PeopleSoft and Other Applications, Second Edition"
]
MongoDB Enterprise >
```

Figura 31. Función distinct con parámetros en MongoDB.

Obtenidos los autores, se va a filtrar por uno de ellos y se van a mostrar los campos «ISBN», «title» y «authors». Nótese que la opción a 0 de ocultado se utiliza solo para el campo `_id` y la opción a 1 para los campos que se quieren mostrar.

```
> db.books.find({authors:"Vlad Landres"},{_id:0, isbn:1, author:1,
title:1})
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.books.find({authors: "Vlad Landres"}, {_id:0, isbn: 1, author: 1, title: 1})
{ "title" : "SQR in PeopleSoft and Other Applications", "isbn" : "188477775" }
{ "title" : "SQR in PeopleSoft and Other Applications, Second Edition", "isbn" : "1932394001" }
MongoDB Enterprise >
```

Figura 32. Función de búsqueda con proyección en MongoDB.

Si se quiere realizar un filtro utilizando los operadores de comparación:

```
> db.books.find({pageCount: {$gt:100} }, {_id:0, isbn:1, author:1, title:1,
pageCount:1}).limit(3)
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.books.find({pageCount: {$gt: 100} }, {_id:0, isbn: 1, author: 1, title: 1, pageCount: 1}).limit(3)
{ "title" : "Unlocking Android", "isbn" : "1933988673", "pageCount" : 416 }
{ "title" : "Android in Action, Second Edition", "isbn" : "1935182722", "pageCount" : 592 }
{ "title" : "Flex 3 in Action", "isbn" : "1933988746", "pageCount" : 576 }
MongoDB Enterprise >
```

Figura 33. Función de búsqueda en MongoDB.

Se puede ver que es muy sencillo realizar una búsqueda con contenido exacto, pero si se quiere hacer una búsqueda con contenido parcial, es necesario utilizar el operador `$regex` (regular expression).

A continuación, se realiza una búsqueda con aquellos títulos que tengan contenida la palabra *Internet*:

```
> db.books.find({title: {$regex: ".*Internet"} }, {_id:0, isbn:1, author:1, title:1})
```

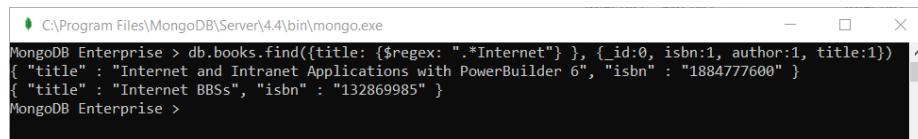
A screenshot of a terminal window titled 'C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe'. The window shows the command 'db.books.find({title: {\$regex: ".*Internet"} }, {_id:0, isbn:1, author:1, title:1})' and its output: two documents. The first document has a title 'Internet and Intranet Applications with PowerBuilder 6', ISBN '1884777600', and author 'MongoDB Enterprise'. The second document has a title 'Internet BB5s', ISBN '132869985', and author 'MongoDB Enterprise'. Both documents have _id:0, isbn:1, and author:1.

Figura 34. Función de búsqueda en MongoDB.

También se puede contar el número de elementos encontrados en la búsqueda con la función `.count()`:

```
> db.books.find({title: {$regex: ".*Internet"} }).count()
```

A screenshot of a terminal window titled 'C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe'. The window shows the command 'db.books.find({title: {\$regex: ".*Internet"} }).count()' and its output: '2'. Both documents found in the previous search are counted.

Figura 35. Función de búsqueda con expresión regular en MongoDB.

Si se quiere insertar un nuevo atributo, se utiliza el operador `$set` como segundo parámetro de un `update`, y si el campo es un `array`, se utiliza el `$addToSet`. A la búsqueda anterior se va a añadir el `tag Internet`. Es muy importante no olvidarse de añadir en el tercer parámetro la opción `multi: true`, para que la modificación se haga en más de un documento.

```
> db.books.update({title: {$regex: ".*Internet"}}, {$addToSet: {tags: "Internet"}}, {multi: true})
> db.books.find({title: {$regex: ".*Internet"} }, {_id:0, isbn:1, author:1, title:1, tags:1})
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.books.update({title: {$regex: ".*Internet"}}, {$addToSet: {tags: "Internet"}}, {multi: true})
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 0 })
MongoDB Enterprise > db.books.find({title: {$regex: ".*Internet"}}, {_id:0, isbn:1, author:1, title:1, tags:1})
[{"title": "Internet and Intranet Applications with PowerBuilder 6", "isbn": "1884777600", "tags": ["Internet"]}, {"title": "Internet BBSS", "isbn": "132869985", "tags": ["Internet"]}]
MongoDB Enterprise >
```

Figura 36. Función de búsqueda con expresión regular en MongoDB.

Se comprueba que todo ha funcionado. Para ello, es necesario realizar una nueva consulta buscando un elemento dentro de un *array*. Para realizar este tipo de búsquedas se utiliza el operador `$in`, el cual realiza una búsqueda por los elementos especificados como parámetros en un *array*, como se puede ver a continuación para aquellos libros que tenga el *tag* Internet:

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.books.find({title: {$regex: ".*Internet"}}, {_id:1, isbn:1, author:1, title:1, tags:1})
[{"_id": 46, "title": "Internet and Intranet Applications with PowerBuilder 6", "isbn": "1884777600", "tags": ["Internet"]}, {"_id": 214, "title": "Internet BBSS", "isbn": "132869985", "tags": ["Internet"]}]
MongoDB Enterprise >
```

Figura 37. Función de búsqueda en MongoDB.

Para volver a dejar todo como estaba en un principio, se utiliza el operador `$unset` para borrar el atributo especificado, que en este caso será `tags`:

```
> db.books.update({}, {$unset: {tags: ""}}, {multi: true})
```

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB Enterprise > db.books.find({title: {$regex: ".*Internet"}}, {_id:1, isbn:1, author:1, title:1, tags:1})
[{"_id": 46, "title": "Internet and Intranet Applications with PowerBuilder 6", "isbn": "1884777600", "tags": ["Internet"]}, {"_id": 214, "title": "Internet BBSS", "isbn": "132869985", "tags": ["Internet"]}]
MongoDB Enterprise > db.books.update({}, {$unset: {tags: ""}}, {multi: true})
WriteResult({ "nMatched" : 451, "nUpserted" : 0, "nModified" : 2 })
MongoDB Enterprise > db.books.find({title: {$regex: ".*Internet"}}, {_id:1, isbn:1, author:1, title:1, tags:1})
[{"_id": 46, "title": "Internet and Intranet Applications with PowerBuilder 6", "isbn": "1884777600"}, {"_id": 214, "title": "Internet BBSS", "isbn": "132869985"}]
MongoDB Enterprise >
```

Figura 38. Función de búsqueda en MongoDB.

Consulta los documentos nuevamente para ver el resultado.

```
> db.books.find({title: {$regex: ".*Internet"}}, {_id:0, isbn:1, author:1, title:1, tags:1})
```

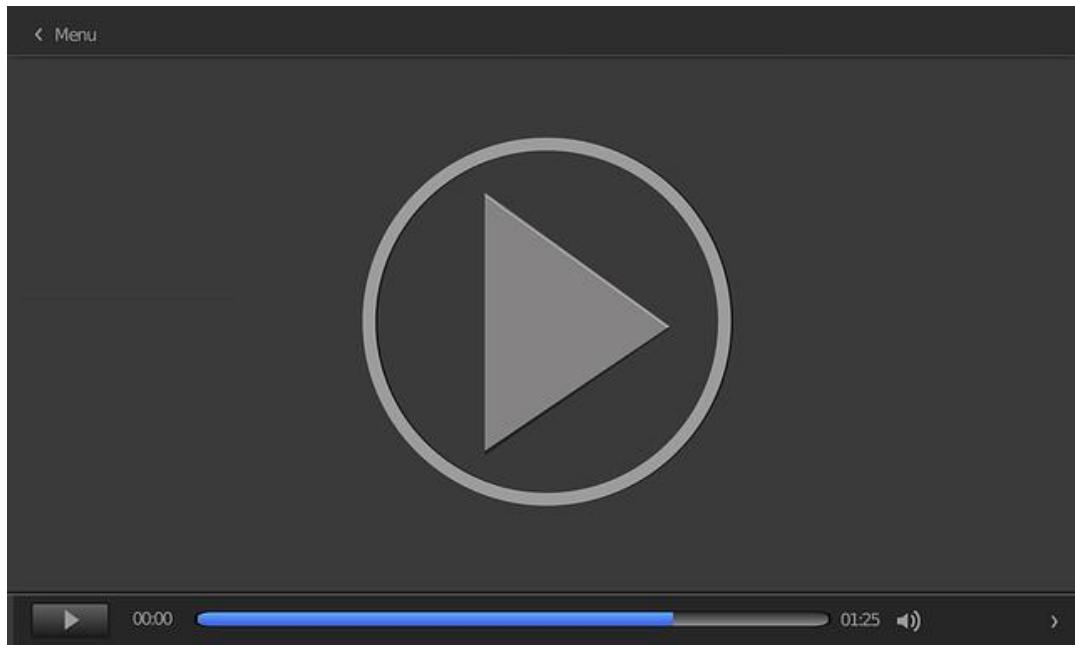
Para terminar, si se quiere realizar una búsqueda en la que se pretende buscar los libros de dos categorías diferentes, es necesario especificarlos dentro de un *array* inicializado por el operador `$or`, como se puede ver a continuación:

```
> db.books.find({$or: [ {categories: "PowerBuilder"}, {categories: "Client-Server"}]}, {_id:0, title:1, isbn:1, categories:1})
```

A screenshot of the MongoDB command-line interface (mongo.exe) window. The command entered is `> db.books.find({$or: [{categories: "PowerBuilder"}, {categories: "Client-Server"}]}, {_id:0, title:1, isbn:1, categories:1})`. The output shows a list of books from the 'books' collection, each with its title, ISBN, and categories. The categories listed are PowerBuilder, Client-Server, Business, Networking, Java, Perl, and Client-Server.

Figura 39. Función de búsqueda en MongoDB.

Después de leer todo el tema, el reto siguiente es que hagas un repaso sobre el uso de MongoDB y sus principales comandos. Para ello, accede al vídeo «Uso de MongoDB» que acompaña este tema.

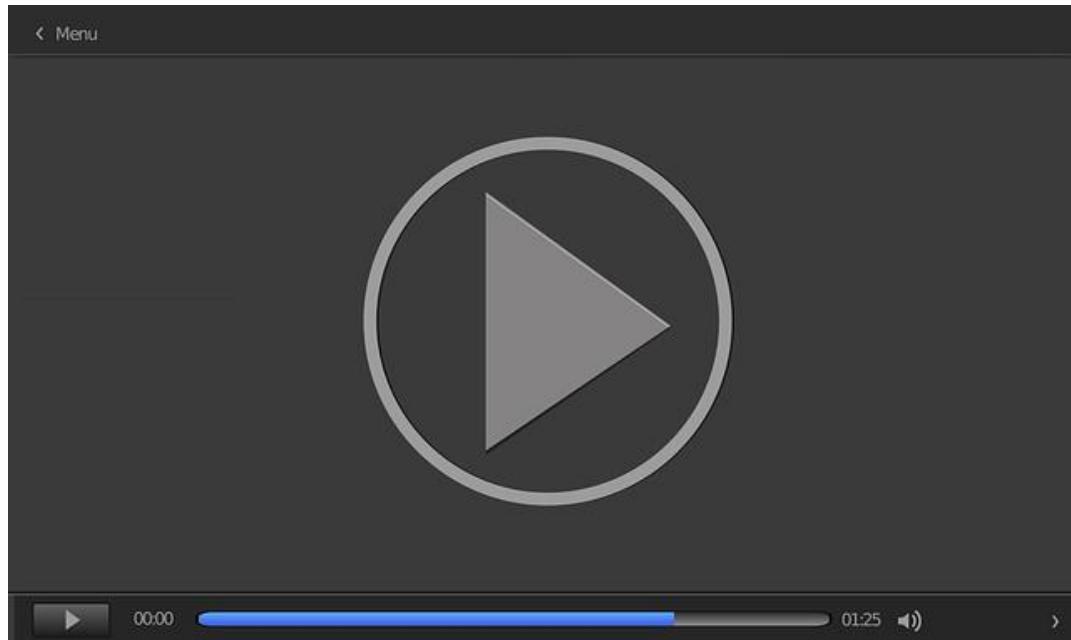


Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=5a533ba3-0c93-46e2-8062-aca701508b4e>

Vídeo 1. Uso de MongoDB.

En esta otra lección titulada «Documentos de MongoDB y patrones de diseño», se presenta un *screencast* de la consola de MongoDB, incluyendo ejemplos de documentos y de patrones de diseño.

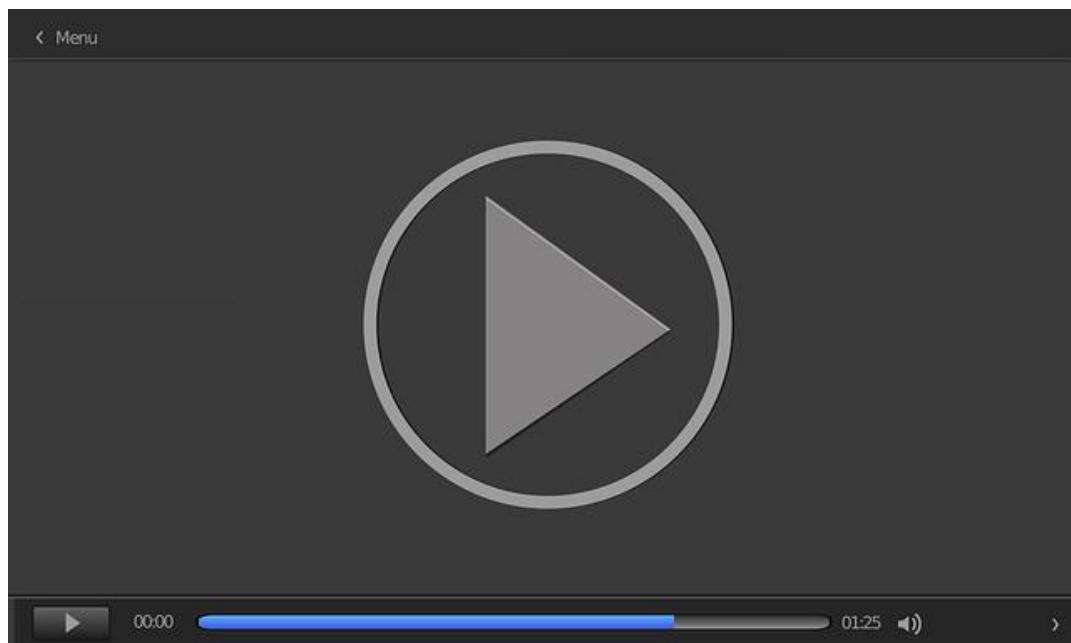


Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=bfe0fc94-a095-4427-9218-adbe00e85197>

Vídeo 2. Documentos de MongoDB y patrones de diseño.

Finalmente, en «Ejemplos de consultas en MongoDB» se presenta un *screencast* sobre varios ejemplos de consultas realizadas en la consola de MongoDB.



Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=96911201-6f53-4e25-8a21-adbe00e865a9>

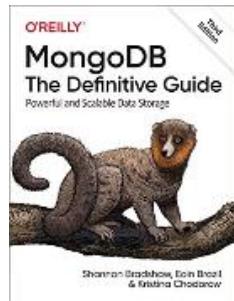
Vídeo 3. Ejemplos de consultas en MongoDB.

3.9. Referencias bibliográficas

MongoDB. (s. f.). *Documentación MongoDB. MongoDB Manual*. Recuperado 14 de junio de 2022, de <https://www.mongodb.com/docs/>

MongoDB: the definitive guide

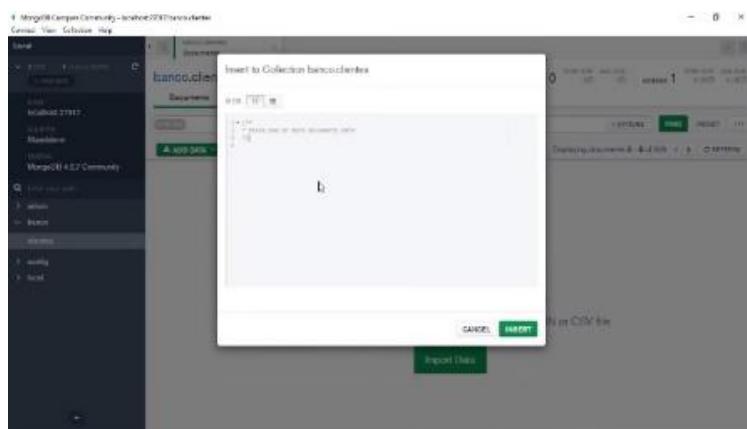
Chodorow, K., y Dirolf, M. (2019). *MongoDB: the definitive guide*. O'Reilly. <https://www.oreilly.com/library/view/mongodb-the-definitive/9781491954454/>



Este libro contiene información detallada sobre la creación, eliminación y modificación de documentos en MongoDB. El capítulo 4 se centra en la realización de consultas.

Bases de datos en MongoDB Compass

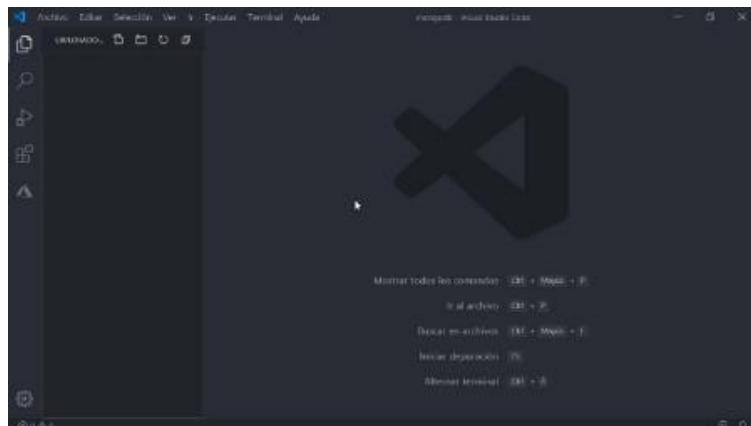
Ávila, J. (22 de junio de 2020). *Como crear Bases de Datos en MongoDB Compass Creación de Colecciones, Documentos y Vistas* [Archivo de vídeo]. <https://www.youtube.com/watch?v=8gbDA7kH5zg>



En este vídeo encontrarás un tutorial muy detallado de operaciones en MongoDB muy usuales, tales como creación de bases de datos utilizando MongoDB Compass, creación de colecciones, documentos, vistas, importar colecciones desde archivos CSV y la forma de establecer agrupación y/o relaciones entre colecciones.

MongoDB: bases de datos, colecciones y documentos

Informática DP. (21 de abril de 2020). 2/3 - *MongoDB 2020 - Bases de datos, Colecciones y Documentos* [Archivo de vídeo]. <https://www.youtube.com/watch?v=Fk5Kq-zhcJo>



En este vídeo encontrarás una guía práctica para aprender a manipular MongoDB desde cero. El IDE utilizado te dará otra visión de herramientas que puedes integrar al uso diario de esta base de datos.

Documentación oficial de MongoDB

MongoDB. (2021). Documentation [Página web]. <http://docs.mongodb.org/>

Dentro del sitio oficial con la documentación del sistema MongoDB, el contenido relevante para este tema es la sección MongoDB CRUD Operations.



A cookbook for MongoDB

MongoDB. (2021). Página web. <http://cookbook.mongodb.org/>

Este sitio web contiene una serie de recetas de soluciones a escenarios comunes en el desarrollo de aplicaciones. Por ejemplo, una de las recetas indica cómo contar los *tags* asociados a *posts* en un sitio web, utilizando el método Map-Reduce.



MongoBooster

MongoDB Admin GUI. (2021). Downloads. *NoSQL Booster* [Página web]. <https://nosqlbooster.com/downloads>

Enlace de descarga, documentación y características de esta GUI para MongoDB.



MongoDB Compass

MongoDB. (2021). MongoDB Compass [Página web]. <https://www.mongodb.com/try/download/compass>

Dentro del sitio oficial se puede acceder a la solución Compass y descargar la aplicación gratuitamente.



Bibliografía

- Banker, K. (2016). *MongoDB in action*. Manning Publications.
- Bradshaw, S., Brazil, E., y Chodorow, K. (2019). *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media.
- Copeland, R. (2013). *MongoDB applied design patterns* (pp. 37-73). O'Reilly Media.
- Dayley, B., Dayley, B., y Dayley, C. (2017). *Node.js, MongoDB and Angular web development: the definitive guide to using the MEAN stack to build web applications*. Addison-Wesley.
- Giamas, A. (2017). *Mastering MongoDB 3.x: an expert's guide to building fault-tolerant MongoDB applications*. Packt Publishing.
- Mukasheva, A., Yedilkhan, D., & Zimin, I. (2021, abril). Uploading Unstructured Data to MONGODB Using the NoSQLBooster Tool. En *2021 IEEE International Conference on Smart Information Systems and Technologies (SIST)* (pp. 1-4). IEEE.
- Tiwari, S. (2011). *Professional NoSQL* (pp. 97-135, 217-232). John Wiley & Sons.

1. ¿Qué comando de la consola de MongoDB se utiliza para indicar la base de datos con la que se trabajará?

- A. select.
- B. find.
- C. use.
- D. Las respuestas A y C son correctas.

2. ¿Cuál de las siguientes afirmaciones es correcta?

- A. MongoBooster es una herramienta GUI multiplataforma que facilita la construcción de consultas.
- B. MongoDB Compass es una herramienta no propietaria para la manipulación externa de bases de datos MongoDB.
- C. MongoBooster y MongoDB Compass proporcionan información estadística y de rendimiento de una base de datos MongoDB.
- D. Todas las afirmaciones anteriores son correctas.

3. ¿Cuál será el resultado al insertar un documento que posee un atributo más al resto de atributos de la colección?

- A. Dará un fallo al insertar los datos porque el modelo de datos es diferente.
- B. Insertará los datos a la colección.
- C. Insertará los datos a la colección y creará el nuevo atributo vacío en el resto de documentos.
- D. Insertará los datos a la colección, pero sin el nuevo atributo para cumplir con el modelo.

4. ¿Qué comando puede utilizarse en MongoDB para la creación de un nuevo documento dentro de una *collection*?
 - A. save.
 - B. insert.
 - C. create.
 - D. Las respuestas A y B son correctas.
5. ¿Cuál es el nombre del atributo especial en las *collections* de MongoDB que ayuda a identificar de manera única a cada documento?
 - A. _id.
 - B. ID.
 - C. Primary key.
 - D. Identifier.
6. ¿Qué método permite modificar los datos de un documento sin tener que incluir el documento completo como argumento?
 - A. save.
 - B. store.
 - C. update.
 - D. set.
7. La operación MongoDB equivalente a JOIN en SQL es:
 - A. Se puede conseguir concatenando sentencias *find* en la misma operación.
 - B. El *aggregation framework*.
 - C. MongoDB no tiene operación equivalente a JOIN hasta su versión 3.2.
 - D. Ninguna de las anteriores es cierta.

- 8.** ¿Qué situación tiene que darse para que el comando `save` actualice un documento?
- A. Que el argumento contenga un identificador existente en la *collection*.
 - B. Que el segundo argumento en el comando sea el valor *true*.
 - C. Que el argumento se parezca en más de un 50 % a un documento en la *collection*.
 - D. `save` no puede utilizarse para actualizar documentos.
- 9.** ¿Qué comando se utiliza en MongoDB para eliminar un conjunto de documentos?
- A. `save`.
 - B. `delete`.
 - C. `remove`.
 - D. `unset`.
- 10.** ¿Qué comando puede aplicarse sobre el resultado de una consulta en MongoDB para restringir el número de documentos retornados?
- A. `limit`.
 - B. `restrict`.
 - C. `skip`.
 - D. `sort`.

Bases de Datos para el Big Data

Agregación

Índice

[Esquema](#)

[Ideas clave](#)

[4.1. Introducción y objetivos](#)

[4.2. Conceptos](#)

[4.3. Map-Reduce](#)

[4.4. Aggregation Framework](#)

[4.5. Casos prácticos](#)

[A fondo](#)

[MongoDB: the definitive guide](#)

[MongoDB Map-Reduce](#)

[MongoDB Aggregation Framework](#)

[Aggregation Framework](#)

[Documentación oficial de MongoDB](#)

[Bibliografía](#)

[Test](#)

AGREGACIÓN		
Concepto	Map-Reduce	Aggregation Framework
Las funciones de agregación son útiles para agrupar datos y obtener resultados a partir de ellos.	Map-Reduce está formada por dos funciones. Una función <i>map</i> que genera los pares clave-valor y otra <i>reduce</i> , que opera sobre ellos.	El framework de agregación está modelado en varias etapas que transforman los documentos en un resultado agregado.
En la base de datos relationales, este tipo de consultas se realizan con el operador group by (para agrupar) y sum , count , etc., para realizar cualquier operación con sus datos.	El lenguaje de programación utilizado es JavaScript.	Las etapas más usadas son la \$match , encargada de la búsqueda, y \$group , la que agrupa y hace los cálculos.
Existen dos métodos de agregación en MongoDB: MapReduce y Aggregate .	Los parámetros de agregación en la función Map-Reduce se especifican como un objeto JSOM .	Es obligatorio especificar el campo _id en el nuevo documento, ya que estos son los valores a agrupar.
	La mayor ventaja de la utilización de este método de agregación es la potencia que ofrece, ya que se pueden personalizar totalmente cada una de sus funciones.	Además de los operadores de búsqueda, tiene muchos otros que hacen la vida más fácil al trabajar, array , string , etc.
		Existen operaciones específicas de agregación: count , distinct y group .

4.1. Introducción y objetivos

Este tema se enfoca en los conceptos básicos de agregación y las dos funcionalidades que tiene MongoDB para llevarlo a cabo. La lectura de las Ideas claves te ayudará a **conocer la sintaxis de los comandos utilizados para realizar la agregación de datos.**

Al tratarse de un tema muy práctico, la mejor forma de estudiarlo es a través de la realización de los trabajos asignados. Además de los trabajos, es recomendable trabajar de forma proactiva con MongoDB, lo cual te ayudará a familiarizarte con el funcionamiento del sistema.

MongoDB permite agregar los datos de una colección a través de operaciones como sumar, contar y otros. A lo largo de este tema, se estudiarán los conceptos básicos de agregación y los dos métodos que utiliza MongoDB para agregar datos: **Map-Reduce** y, a partir de la versión 2.2, **Aggregate**. En especial, se estudiará el segundo por su importancia actual.

Los objetivos que se buscan alcanzar con este tema son los siguientes:

- ▶ Comprender los métodos Map-Reduce y Aggregate.
- ▶ Diferenciar los métodos y saber aplicar uno u otro en función del caso de uso.
- ▶ Poner en práctica el uso de ambos métodos a través de ejercicios guiados.

4.2. Conceptos

Cualquier persona que trabaje con bases de datos tiene que realizar con frecuencia consultas para agrupar datos y obtener resultados a partir de ellas. Por ejemplo, si se quiere conocer la suma total de ventas que cada comercial de una empresa ha conseguido en un año, es necesario sumar cada una de las ventas que se han hecho por comercial y agruparlas por el nombre de ese comercial.

Al final, se obtiene una tabla en la que aparecen comerciales con la suma de sus ventas en el período marcado en la consulta, que en este caso sería de un año.

En las bases de datos relacionales, este tipo de consultas se realizan con el operador **group by** (para agrupar) y **sum**, **count**, etc. para realizar cualquier operación con sus datos. MongoDB dispone de tres métodos para gestionar la agregación: Map-Reduce, Aggregation Framework y operadores simples de agregación específicos.

Map-Reduce es un modelo de programación utilizado por Google para el cálculo en paralelo de grandes volúmenes de datos. Debido a la incursión del *big data*, este modelo ha tomado mucha fuerza y se ha añadido a Framework y tecnologías como MongoDB, Hadoop y muchas otras.

Por el contrario, **MongoDB incorporó a partir de su versión 2.2 Aggregation Framework**, para poder realizar **operaciones de agregación** similares a las que se hacen en los lenguajes **SQL relacionales**.

Aunque Map-Reduce es mucho más potente, ya que se tiene todo un lenguaje de programación para realizar cálculos y generar salidas, también es más difícil de

utilizar por el nivel de programación que este requiere. Por ello, lo más recomendable es utilizar Aggregation Framework para realizar cálculos tanto sencillos como más complejos.

Otra de las razones por las que se recomienda más utilizar Aggregation Framework es porque en temas de rendimiento está mucho más optimizado y, por lo tanto, el tiempo de la consulta suele ser mucho menor.

Además de los dos modelos de agregación, MongoDB proporciona dos operadores simples que no suelen incluirse dentro de los métodos de agregación descritos: count y distinct. El primero cuenta el número de resultados obtenidos a partir de la búsqueda y el segundo muestra los resultados que son distintos del campo que se especifique como parámetro. A continuación, se puede ver el formato de distinct :

```
db.collection.distinct(<atributo>, {<filtgro>})
```

Donde el atributo es el campo donde se hace el distinct y el filtro, los parámetros de la búsqueda. Un ejemplo real sería:

```
db.coches.distinct("modelo", {marca: "bmw"})
```

4.3. Map-Reduce

Las operaciones de *map-reduce* tienen dos fases:

- ▶ Una etapa de *map* que procesa cada documento y emite uno o más objetos para cada documento de entrada.
- ▶ Una fase de *reduce* que combina la salida de la operación del *map*.

Opcionalmente, Map-Reduce puede tener una etapa de finalización para realizar modificaciones finales al resultado. También puede especificar una condición de consulta para seleccionar los documentos de entrada, así como ordenar y limitar los resultados.

Map-Reduce utiliza funciones de JavaScript personalizadas para realizar el *map* y el *reduce* de las operaciones, así como la operación de finalización opcional. El uso de JavaScript proporciona una gran flexibilidad en comparación con las *pipelines*, pero Map-Reduce es menos eficiente y más complejo.

En el siguiente recuadro se muestra la sintaxis del método Map-Reduce:

El método Map-Reduce hace uso de una función *map*, la cual procesa un conjunto de documentos y, mediante el comando , retorna un par clave-valor, utilizando los atributos de cada documento. Todos los valores de cada par clave-valor son agrupados de forma en que cada clave tendrá asociado un *array* con los valores apropiados.

La función *reduce* recibe la clave y el *array* de valores como primer y segundo

argumento respectivamente; la tarea de esta función es realizar operaciones de agregación sobre los valores del *array* y retornar el valor resultante. Este valor será asociado a la clave respectiva.

Los parámetros de agregación en la función Map-Reduce se especifican como un objeto JSON, con los siguientes atributos:

- ▶ **out**: es el único atributo obligatorio. Indica el nombre de la *collection* donde se almacenará el resultado de la operación Map-Reduce. También puede indicarse la función `return` el resultado directamente, a través del objeto `_`.
- ▶ **query**: permite definir las condiciones de búsqueda para filtrar los documentos que serán procesados.
- ▶ **sort**: indica la forma en la que se ordenarán los documentos antes de ser procesados, lo cual permite optimizar el proceso.
- ▶ **limit**: indica el número máximo de documentos a retornar.
- ▶ **finalize**: es una función opcional para aplicar sobre los documentos retornados por la función *reduce*.
- ▶ **scope**: indica variables globales que serán accesibles desde las funciones *map*, *reduce* y *finalize*.
- ▶ **jsMode**: valor booleano para indicar si los documentos deben convertirse a formato JSON en vez de BSON entre las funciones *map* y *reduce*. Por defecto es false.
- ▶ **verbose**: valor booleano que indica si se debe mostrar información sobre el tiempo de procesado de la función Map-Reduce. El valor por defecto es true.

La siguiente captura muestra un ejemplo de ejecución del comando `aggregate` para calcular la suma los precios de todos los documentos en la *collection* *productos*.

```
> db.productos.mapReduce(  
...  function() { emit("Suma", this.precio) },  
...  function(key, values) { return Array.sum(values) },  
...  { out: {inline: 1} }  
... )  
{  
    "results" : [  
        {  
            "_id" : "Suma",  
            "value" : 8.95  
        }  
    ],  
    "timeMillis" : 0,  
    "counts" : {  
        "input" : 5,  
        "emit" : 5,  
        "reduce" : 1,  
        "output" : 1  
    },  
    "ok" : 1,  
}  
>
```

Figura 1. Ejemplo de Map-Reduce en la consola de MongoDB.

En el ejemplo, la función *map* emite pares clave-valor con la cadena de texto «Suma» como clave constante y el atributo *precio* como el valor. La función *reduce* se encarga de sumar los valores del *array* de precios, y este valor se asigna a la clave «Suma» indicada previamente.

Además, se indica que la función retornará directamente los valores mediante el atributo `.`. Los resultados del método Map-Reduce se pueden obtener como un *array* de documentos dentro del atributo `.`.

En el siguiente ejemplo, se va realizar una operación Map-Reduce más compleja, en la colección de pedidos para todos los documentos que tengan un valor `date` mayor que 01/01/2012. La operación agrupa por el campo `sku` y calcula el número de pedidos y la cantidad total para cada `sku`. La operación concluye calculando la cantidad media por pedido para cada valor de `sku`.

Si se parte del siguiente formato de documento:

```
{  
  _id: ObjectId("50a8240b927d5d8b5891743c"),  
  cust_id: "abc123",  
  ord_date: new Date("Oct 04, 2012"),  
  status: 'A',  
  price: 25,  
  items: [ { sku: "mmm", qty: 5, price: 2.5 },  
           { sku: "nnn", qty: 5, price: 2.5 } ]  
}
```

La función *map* que procesa cada documento:

- ▶ En la función, `this` se refiere al documento que la operación Map-Reduce está procesando.
- ▶ Por cada elemento, la función asocia `this` con el nuevo objeto `value` que contiene el campo `count` a 1 y el elemento `sku` para el perdido, y emite el par `(key, value)`.

```
var mapFunction = function() {  
  for (var idx = 0; idx < this.items.length; idx++) {  
    var key = this.items[idx].sku;  
    var value = {  
      count: 1,  
      qty: this.items[idx].qty  
    };  
    emit(key, value);  
  }  
};
```

Se define la función *reduce* con dos argumentos: `keySKU` y `countObjVals`:

- ▶ `keySKU` es un *array* cuyos elementos son los objetos asignados a los valores de agrupados pasados por la función *map* a la función *reduce*.
- ▶ La función *reduce* la matriz `value` a un solo objeto `value` que contiene los campos `count` y `total_qty`.

- ▶ En `key`, el campo `count` contiene la suma de los campos `count` de los elementos individuales del `array`, y el campo `qty` contiene la suma de los campos `qty` de los elementos individuales del `array`:

```
var reduceFunction = function(keySKU, countObjVals) {  
    reducedVal = { count: 0, qty: 0 };  
    for (var idx = 0; idx < countObjVals.length; idx++) {  
        reducedVal.count += countObjVals[idx].count;  
        reducedVal.qty += countObjVals[idx].qty;  
    }  
    return reducedVal;  
};
```

Para terminar, se define una función de finalización con dos argumentos `key` y `reducedVal`. La función modifica el objeto `reducedVal` para agregar un campo computado llamado `avg` y devuelve el objeto modificado:

```
var finalizeFunction = function (key, reducedVal) {  
    reducedVal.avg = reducedVal.qty/reducedVal.count;  
    return reducedVal;  
};
```

Para terminar, se realiza la operación de Map-Reduce en la colección utilizando todas las funciones anteriores:

```
db.orders.mapReduce( mapFunction, reduceFunction, {  
    out: { merge: "map_reduce_example" },  
    query:{ ord_date: { $gt: new Date('01/01/2012') } },  
    finalize: finalizeFunction  
});
```

Esta operación utiliza el campo de consulta para seleccionar solo aquellos documentos con `ord_date` mayor que la fecha 01/01/2012. Seguidamente, emite los resultados a una colección `map_reduce_example`. Si ya existe la colección `map_reduce_example`, la operación combinará el contenido existente con los resultados de esta operación.

Ten en cuenta que:

- ▶ A partir de MongoDB 2.4, ciertas funciones y propiedades de la consola son inaccesibles en las operaciones de reducción de mapas.
- ▶ MongoDB 2.4 también proporciona soporte para múltiples operaciones de JavaScript para ejecutarse al mismo tiempo.
- ▶ Antes de MongoDB 2.4, el código JavaScript se ejecutaba en un solo hilo, lo que generaba problemas de concurrencia para reducir el mapa.
- ▶ A partir de la versión 4.2, se depreca la opción de Map-Reduce para crear una nueva colección fragmentada, así como el uso de la opción de fragmentación para Map-Reduce. Para enviar a una colección fragmentada, crea primero la colección fragmentada.
- ▶ MongoDB 4.2 también depreca el reemplazo de una colección fragmentada existente y la especificación explícita de `nonAtomic: false`.

4.4. Aggregation Framework

El *framework* de agregación de MongoDB está modelado en el concepto de tuberías de procesamiento de datos, es decir, los documentos entran en una tubería (*pipeline*) de varias etapas que transforman los documentos en un resultado agregado.

Las etapas más básicas proporcionan filtros que funcionan como consultas y transformaciones de documentos que modifican la forma del documento de salida.

El resto de operaciones proporcionan herramientas para agrupar y clasificar documentos por campo o campos específicos, así como herramientas para agregar el contenido de *arrays*, incluyendo *arrays* de documentos. Además, las etapas pueden utilizar operadores para tareas tales como calcular el promedio o concatenación de cadenas.

Aggregate funciona perfectamente con la capacidad de *sharding* que proporciona MongoDB y que se verá más adelante. Además, se puede utilizar índices para mejorar el rendimiento durante algunas de sus etapas. Independientemente, Aggregate tiene una fase de optimización interna, lo que le permite mejorar sustancialmente en rendimiento a Map-Reduce.

A continuación, se ilustra el funcionamiento de Aggregate con un ejemplo básico:

```
db.coches.aggregate( [  
    $match stage ->{$match: {marca: "BMW" } },  
    $group stage -> {$group: {_id: "$modelo", total: { $sum: "$precio"} } },  
    more stage -> { ... },  
    more stage -> { ... }  
]  
)
```

El funcionamiento en etapas sería el siguiente:

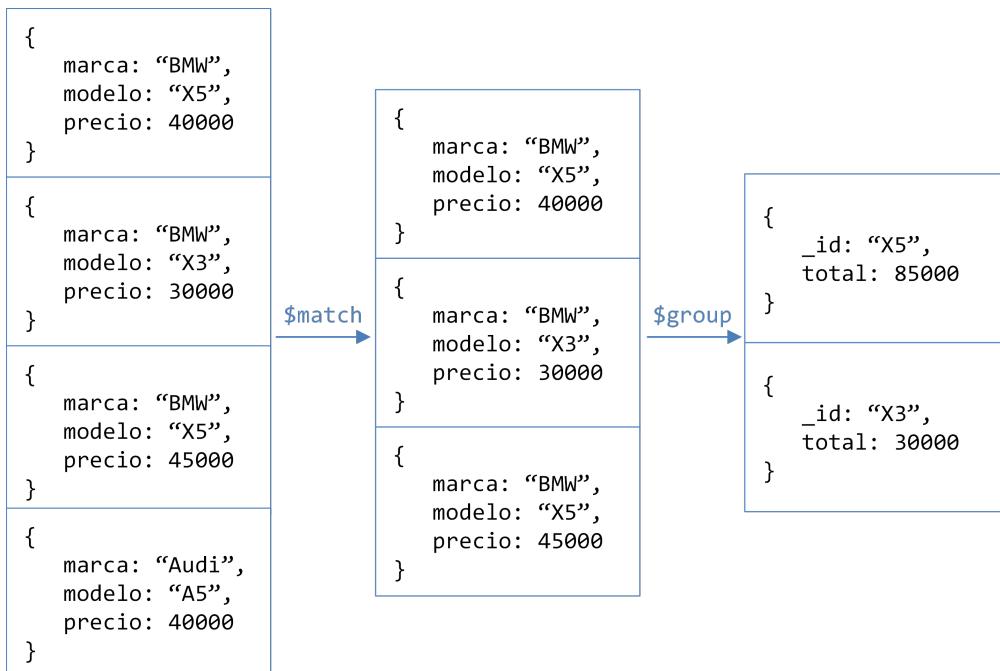


Figura 2. Ejemplo de agrupación con Aggregate en MongoDB.

Como se puede ver, la etapa de filtrado (\$smatch) es análoga a la vista en los filtros convencionales de MongoDB, por lo que esta operación busca los documentos según los valores especificados.

Por otro lado, la etapa de agrupación (\$group), que es quizá la más usada en el proceso de agrupación, permite agrupar y realizar los cálculos sobre los documentos. Nótese que es obligatorio especificar el campo `_id` en el nuevo documento, ya que estos son los valores para agrupar.

Es posible definir agrupaciones para campos múltiples. Esta etapa trasforma el documento de salida, por lo que en ella se deben especificar, como se ha podido ver, los nuevos atributos de la colección y cómo se obtienen estos atributos. MongoDB permite utilizar los siguientes operadores:

- ▶ **\$min**: devuelve el valor mínimo de los valores numéricos.
- ▶ **\$max**: devuelve el valor máximo de los valores numéricos.

- ▶ **\$avg**: devuelve el valor medio de los valores numéricos.
- ▶ **\$sum**: calcula y devuelve la suma de los valores numéricos.
- ▶ **\$first**: devuelve el valor que resulta de aplicar una expresión al primer documento de un grupo de documentos que comparten el mismo grupo por un campo. Solo tiene sentido cuando los documentos están en un orden definido. Un ejemplo de uso es en fechas.
- ▶ **\$last**: devuelve el valor que resulta de aplicar una expresión al último documento de un grupo de documentos que comparten el mismo grupo por un campo. Solo tiene sentido cuando los documentos están en un orden definido.
- ▶ **\$addToSet**: devuelve un *array* con todos los valores únicos que resulta de aplicar una expresión a cada documento de un grupo de documentos que comparten el mismo grupo por clave. El orden de los elementos en la matriz de salida no está especificado.
- ▶ **\$push**: devuelve un *array* con todos los valores que resultan de la aplicación de una expresión a cada documento de un grupo de documentos que comparten el mismo grupo por clave.
- ▶ **\$stdDevPop**: (desde la versión 3.4) calcula la desviación estándar de la población de los valores de entrada. Se debe utilizar si los valores abarcan toda la población de datos que desea representar y no porque se deseé generalizar sobre una población mayor. Omite valores no numéricos.
- ▶ **\$stdDevSamp**: (desde la versión 3.4.) Calcula la desviación estándar de la muestra de los valores de entrada. Se debe utilizar si los valores abarcan una muestra de una población de datos a partir de la cual se generaliza sobre la población total. Omite valores no numéricos.

El *framework* de agregación permite definir un mayor número de etapas que las dos

que se han descrito anteriormente. El esquema general, por lo tanto, es el siguiente:

```
db.collection.aggregate( [ { <etapa1> }, { <etapa2> }, ... ] )
```

A continuación, se detallan todas las etapas posibles:

- ▶ **\$collStats**: devuelve estadísticas sobre una colección o una vista.
- ▶ **\$project**: redimensiona cada documento en el flujo, por ejemplo, agregando nuevos campos o eliminando campos existentes. Para cada documento de entrada, se genera un documento.
- ▶ **\$match**: filtra el flujo de documentos para permitir que solo los documentos coincidentes pasen sin modificaciones a la siguiente fase de la canalización.
- ▶ **\$redact**: redimensiona cada documento en el flujo restringiendo el contenido de cada documento basado en la información almacenada en los propios documentos. Incorpora la funcionalidad de `$project` y `$smatch`. Puede utilizarse para implementar la redacción a nivel de campo. Para cada documento de entrada, genera uno o cero documentos.
- ▶ **\$limit**: transmite los primeros n documentos no modificados a la canalización donde n es el límite especificado. Para cada documento de entrada, se obtiene un documento (para los primeros n documentos) o cero documentos (después de los primeros n documentos).
- ▶ **\$skip**: salta los primeros n documentos donde n es el número de salto especificado y pasa los documentos restantes sin modificar a la tubería. Para cada documento de entrada, se emiten documentos cero (para los primeros n documentos) o un documento (después de los primeros n documentos).
- ▶ **\$unwind**: destruye un campo de un `array` de los documentos de entrada para generar un documento para cada elemento. Cada documento de salida reemplaza el `array` con un valor de elemento. Para cada documento de entrada, se producen n documentos donde n es el número de elementos del `array` y puede ser cero para

una matriz vacía.

- ▶ **\$group:** agrupa los documentos de entrada por una expresión de identificador especificada y aplica la(s) expresión(es) acumuladora(s), si se especifica, a cada grupo. Consumo todos los documentos de entrada y genera un documento por cada grupo distinto. Los documentos de salida solo contienen el campo identificador y, si se especifica, los campos acumulados.
- ▶ **\$sample:** selecciona aleatoriamente el número especificado de documentos de su entrada.
- ▶ **\$sort:** reordena el flujo de documentos mediante una clave de clasificación especificada. Solo cambia el orden; Los documentos permanecen sin modificar. Para cada documento de entrada, se genera un documento.
- ▶ **\$geoNear:** devuelve un flujo ordenado de documentos basado en la proximidad a un punto geoespacial. Incorpora la funcionalidad de \$match, \$sort y \$limit para los datos geoespaciales. Los documentos de salida incluyen un campo de distancia adicional y pueden incluir un campo de identificador de ubicación.
- ▶ **\$lookup:** realiza una combinación externa izquierda con otra colección en la misma base de datos para filtrar en documentos de la colección «unida» para su procesamiento.
- ▶ **\$out:** escribe los documentos resultantes de la tubería de agregación en una colección. Para usar la etapa \$out , debe ser la última etapa en la tubería.
- ▶ **\$indexStats:** devuelve estadísticas sobre el uso de cada índice para la colección.
- ▶ **\$facet:** procesa múltiples tuberías de agregación dentro de una sola etapa en el mismo conjunto de documentos de entrada. Permite la creación de agregaciones multifacéticas capaces de caracterizar datos a través de múltiples dimensiones, o facetas, en una sola etapa.

- ▶ **\$bucket**: categoriza los documentos entrantes en grupos, llamados cubos, basados en una expresión especificada y límites de cubo.
- ▶ **\$bucketAuto**: clasifica los documentos entrantes en un número específico de grupos, llamados cubos, en función de una expresión especificada. Los límites del cubo se determinan automáticamente en un intento de distribuir uniformemente los documentos en el número especificado de cubos.
- ▶ **\$sortByCount**: agrupa los documentos entrantes basados en el valor de una expresión especificada y, a continuación, calcula el recuento de documentos en cada grupo distinto.
- ▶ **\$addFields**: agrega nuevos campos a los documentos. Produce documentos que contienen todos los campos existentes de los documentos de entrada y campos añadidos recientemente.
- ▶ **\$replaceRoot**: reemplaza un documento con el documento incrustado especificado. La operación reemplaza todos los campos existentes en el documento de entrada, incluido el campo `_id`. Especifica un documento incrustado en el documento de entrada para promocionar el documento incrustado al nivel superior.
- ▶ **\$count**: devuelve un recuento del número de documentos en esta etapa de la canalización de agregación.
- ▶ **\$graphLookup**: realiza una búsqueda recursiva en una colección. A cada documento de salida, agrega un nuevo campo de matriz que contiene los resultados de recorrido de la búsqueda recursiva para ese documento.

Los operadores de expresión se construyen para construir expresiones en algunas de las etapas de la tubería. Algunos de ellos ya se han visto en el tema anterior. El formato es el siguiente:

```
{<operador>: [ <argumento1>, <argumento2>, ... ]}
```

A continuación, se enumera cada tipo de operador:

- ▶ **Booleanos:** \$and , \$or , \$not
- ▶ **De conjunto:** \$setEquals , \$setIntersection , \$setUnion , \$setDifference , \$setIsSubset ,
\$anyElementTrue , \$allElementsTrue
- ▶ **De comparación:** \$cmp , \$eq , \$gt , \$gte , \$lt , \$lte , \$ne
- ▶ **Aritméticos:** \$abs , \$add , \$ceil , \$divide , \$exp , \$floor , \$ln , \$log , \$log10 , \$mod ,
\$multiply , \$pow , \$sqrt , \$subtract , \$trunc
- ▶ **De cadena de caracteres:** \$concat , \$indexOfBytes , \$indexOfCP , \$split , \$strLenBytes ,
\$strLenCP , \$strcasecmp , \$substr , \$substrBytes , \$substrCP , \$toLower, \$toUpper .
- ▶ **Búsqueda de texto:** \$meta
- ▶ **Array:** \$arrayElemAt , \$concatArrays , \$filter , \$indexFromArray , \$isArray, \$range , \$reverseArray
, \$reduce , \$size , \$slice , \$zip , \$in
- ▶ **Variable:** \$map , \$let
- ▶ **Literal:** \$literal
- ▶ **Date:** \$dayOfYear , \$dayOfMonth , \$dayOfWeek , \$year , \$month , \$week , \$hour , \$minute ,
\$second , \$millisecond , \$dateToString , \$isoDayOfWeek , \$isoWeek , \$isoWeekYear
- ▶ **Condicional:** \$cond , \$ifNull , \$switch
- ▶ **Tipo de datos:** \$type

4.5. Casos prácticos

En este apartado pondrás en práctica lo aprendido con el método Map-Reduce y lo compararás con las funciones de Aggregate. Se proponen tres casos prácticos donde debes:

- ▶ Insertar los datos.
- ▶ Crear las funciones Map-Reduce.
- ▶ Escribir la función Aggregate.
- ▶ Finalmente, consultar el documento de salida de las operaciones anteriores.

Los datos del primer y segundo caso no son iguales a los datos propuestos para el tercer caso.

Caso 1: usar Map-Reduce y comparar con Aggregate.

- ▶ Para los siguientes ejercicios, utiliza el siguiente conjunto de documentos de la colección `norders`.

```
<pre><code>db.norders.insertMany([
  { _id: 1, cust_id: "Ant O. Knee", ord_date: new Date("2020-03-01"), price: 25, items:
    [ { sku: "oranges", qty: 5, price: 2.5 }, { sku: "apples", qty: 5, price: 2.5 } ], status: "A" },
  { _id: 2, cust_id: "Ant O. Knee", ord_date: new Date("2020-03-08"), price: 70, items:
    [ { sku: "oranges", qty: 8, price: 2.5 }, { sku: "chocolates", qty: 5, price: 10 } ], status: "A" },
  { _id: 3, cust_id: "Busby Bee", ord_date: new Date("2020-03-08"), price: 50, items:
    [ { sku: "oranges", qty: 10, price: 2.5 }, { sku: "pears", qty: 10, price: 2.5 } ], status: "A" },
  { _id: 4, cust_id: "Busby Bee", ord_date: new Date("2020-03-18"), price: 25, items:
    [ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },
  { _id: 5, cust_id: "Busby Bee", ord_date: new Date("2020-03-19"), price: 50, items:
    [ { sku: "chocolates", qty: 5, price: 10 } ], status: "A" },
  { _id: 6, cust_id: "Cam Elot", ord_date: new Date("2020-03-19"), price: 35, items:
    [ { sku: "carrots", qty: 10, price: 1.0 }, { sku: "apples", qty: 10, price: 2.5 } ], status: "A" },
])</code></pre>
```

```
{ _id: 7, cust_id: "Cam Elot", ord_date: new Date("2020-03-20"), price: 25, items:  
[ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },  
{ _id: 8, cust_id: "Don Quis", ord_date: new Date("2020-03-20"), price: 75, items:  
[ { sku: "chocolates", qty: 5, price: 10 }, { sku: "apples", qty: 10, price: 2.5 } ], status: "A" },  
{ _id: 9, cust_id: "Don Quis", ord_date: new Date("2020-03-20"), price: 55, items:  
[ { sku: "carrots", qty: 5, price: 1.0 }, { sku: "apples", qty: 10, price: 2.5 },  
{ sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },  
{ _id: 10, cust_id: "Don Quis", ord_date: new Date("2020-03-23"), price: 25, items:  
[ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" }  
])
```

- ▶ Escribe las siguientes funciones:

```
<pre><code>var mapFunction1 = function() {  
    emit(this.cust_id, this.price);  
};  
var reduceFunction1 = function(keyCustId, valuesPrices) {  
    return Array.sum(valuesPrices);  
};
```

- ▶ Ahora aplica las funciones sobre los documentos anteriores:

```
<pre><code>db.norders.mapReduce(  
    mapFunction1,  
    reduceFunction1,  
    { out: "map_reduce_example" }  
)
```

- ▶ Observa que el resultado de Map-Reduce se almacena en una nueva colección llamada `map_reduce_example`. El paso siguiente es ejecutar una consulta sobre esta nueva colección.

```
<pre><code>db.map_reduce_example.find().sort({_id:1})  
<{ "_id" : "Ant O. Knee", "value" : 95 }>  
<{ "_id" : "Busby Bee", "value" : 125 }>
```

```
{ "_id" : "Cam Elot", "value" : 60 }  
{ "_id" : "Don Quis", "value" : 155 }
```

- ▶ Ahora realiza la misma operación anterior, pero utilizando la función `aggregate`.

```
<pre><code>db.norders.aggregate([  
  { $group: { _id: "$cust_id", value: { $sum: "$price" } } },  
  { $out: "agg_alternative_1" }  
)
```

- ▶ Repite la última consulta para comprobar el resultado.

```
<pre><code>  
db.agg_alternative_1.find().sort({_id:1})
```

- ▶ Puedes observar la diferencia que hay entre el uso de Map-Reduce y Aggregate.

Caso 2: utiliza Map-Reduce y compara con Aggregate, añade además la cláusula

finalize

- ▶ Escribe las siguientes funciones:

```
<pre><code>var mapFunction2 = function() {  
  for (var idx = 0; idx < this.items.length; idx++) {  
    var key = this.items[idx].sku;  
    var value = { count: 1, qty: this.items[idx].qty };  
    emit(key, value);  
  }  
};  
  
var reduceFunction2 = function(keySKU, countObjVals) {  
  reducedVal = { count: 0, qty: 0 };  
  for (var idx = 0; idx < countObjVals.length; idx++) {  
    reducedVal.count += countObjVals[idx].count;  
    reducedVal.qty += countObjVals[idx].qty;  
  }  
}
```

```
return reducedVal;
};

var finalizeFunction2 = function (key, reducedVal) {
    reducedVal.avg = reducedVal.qty/reducedVal.count;
    return reducedVal;
};
```

- ▶ Ahora aplica las funciones sobre los documentos anteriores:

```
<pre><code>db.norders.mapReduce(  
    mapFunction2,  
    reduceFunction2,  
    {  
        out: { merge: "map_reduce_example2" },  
        query: { ord_date: { $gte: new Date("2020-03-01") } },  
        finalize: finalizeFunction2  
    }  
>);
```

- ▶ Recuerda consultar la nueva colección producto del Map-Reduce creado previamente.

```
<pre><code>db.map_reduce_example2.find().sort( { _id: 1 } )
```

- ▶ Ahora realiza la misma operación anterior, pero utilizando la función `aggregate`.

```
<pre><code>db.norders.aggregate( [  
    { $match: { ord_date: { $gte: new Date("2020-03-01") } } },  
    { $unwind: "$items" },  
    { $group: { _id: "$items.sku", qty: { $sum: "$items.qty" }, orders_ids: { $addToSet: "$_id" } } },  
    { $project: { value: { count: { $size: "$orders_ids" }, qty: "$qty", avg: { $divide: [ "$qty", { $size: "$orders_ids" } ] } } } },  
    { $merge: { into: "agg_alternative_3", on: "_id", whenMatched: "replace", whenNotMatched: "insert" } }  
>)
```

- ▶ Repite la última consulta para comprobar el resultado.

```
<pre><code>db.agg_alternative_3.find().sort( { _id: 1 } )
```

Caso 3: utiliza Map-Reduce y compara con Aggregate, añade además la cláusula

finalize

y observa la agrupación incremental

- ▶ Para el siguiente ejercicio, utiliza el siguiente conjunto de documentos de la colección `usersessions`.

```
<pre><code>db.usersessions.insertMany([
    { userid: "a", start: ISODate('2020-03-03 14:17:00'), length: 95 },
    { userid: "b", start: ISODate('2020-03-03 14:23:00'), length: 110 },
    { userid: "c", start: ISODate('2020-03-03 15:02:00'), length: 120 },
    { userid: "d", start: ISODate('2020-03-03 16:45:00'), length: 45 },
    { userid: "a", start: ISODate('2020-03-04 11:05:00'), length: 105 },
    { userid: "b", start: ISODate('2020-03-04 13:14:00'), length: 120 },
    { userid: "c", start: ISODate('2020-03-04 17:00:00'), length: 130 },
    { userid: "d", start: ISODate('2020-03-04 15:37:00'), length: 65 }
])
db.usersessions.insertMany([
    { userid: "a", ts: ISODate('2020-03-05 14:17:00'), length: 130 },
    { userid: "b", ts: ISODate('2020-03-05 14:23:00'), length: 40 },
    { userid: "c", ts: ISODate('2020-03-05 15:02:00'), length: 110 },
    { userid: "d", ts: ISODate('2020-03-05 16:45:00'), length: 100 }
])</code></pre>
```

- ▶ Escribe las siguientes funciones:

```
<pre><code>var mapFunction = function() {
    var key = this.userid;
    var value = { total_time: this.length, count: 1, avg_time: 0 };
    emit( key, value );
};

var reduceFunction = function(key, values) {
    var reducedObject = { total_time: 0, count: 0, avg_time: 0 };
    values.forEach(function(value) {
        reducedObject.total_time += value.total_time;
        reducedObject.count++;
        reducedObject.avg_time = reducedObject.total_time / reducedObject.count;
    });
    return reducedObject;
}</code></pre>
```

```
reducedObject.count += value.count;
});
return reducedObject;
};

var finalizeFunction = function(key, reducedValue) {
if (reducedValue.count > 0)
reducedValue.avg_time = reducedValue.total_time / reducedValue.count;
return reducedValue;
};
```

- ▶ Ahora aplica las funciones sobre los documentos anteriores:

```
<pre><code>db.usersessions.mapReduce(  
    mapFunction,  
    reduceFunction,  
    {  
        out: "session_stats",  
        finalize: finalizeFunction  
    }  
)  
db.session_stats.find().sort( { _id: 1 } )
```

- ▶ Observa el resultado obtenido.

```
<pre><code>{ "_id": "a", "value": { "total_time": 330, "count": 3, "avg_time": 110 } }  
{ "_id": "b", "value": { "total_time": 270, "count": 3, "avg_time": 90 } }  
{ "_id": "c", "value": { "total_time": 360, "count": 3, "avg_time": 120 } }  
{ "_id": "d", "value": { "total_time": 210, "count": 3, "avg_time": 70 } }
```

- ▶ Prueba ahora la siguiente forma y observa el resultado.

```
<pre><code>db.usersessions.mapReduce(  
    mapFunction,  
    reduceFunction,  
    {  
        query: { ts: { $gte: ISODate('2020-03-05 00:00:00') } },  
        out: { reduce: "session_stats" },  
        finalize: finalizeFunction  
    }  
)  
db.session_stats.find().sort( { _id: 1 } )
```

- ▶ Realiza la misma tarea, pero ahora con `aggregate`. Utiliza el siguiente conjunto de documentos de la colección `usersessions`.

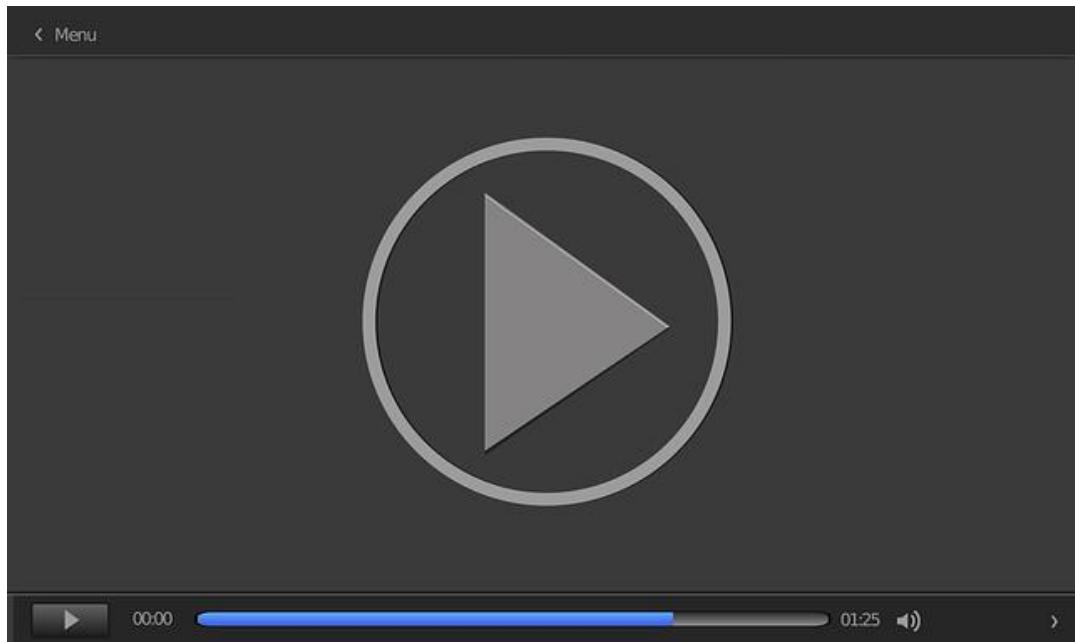
```
<pre><code>db.usersessions.drop();
db.usersessions.insertMany([
  { userid: "a", start:ISODate('2020-03-03 14:17:00'), length: 95 },
  { userid: "b", start:ISODate('2020-03-03 14:23:00'), length: 110 },
  { userid: "c", start:ISODate('2020-03-03 15:02:00'), length: 120 },
  { userid: "d", start:ISODate('2020-03-03 16:45:00'), length: 45 },
  { userid: "a", start:ISODate('2020-03-04 11:05:00'), length: 105 },
  { userid: "b", start:ISODate('2020-03-04 13:14:00'), length: 120 },
  { userid: "c", start:ISODate('2020-03-04 17:00:00'), length: 130 },
  { userid: "d", start:ISODate('2020-03-04 15:37:00'), length: 65 }
])
db.usersessions.aggregate([
  { $group: { _id: "$userid", total_time: { $sum: "$length" }, count: { $sum: 1 }, avg_time: { $avg: "$length" } } },
  { $project: { value: { total_time: "$total_time", count: "$count", avg_time: "$avg_time" } } },
  { $merge: {
    into: "session_stats_agg",
    whenMatched: [ { $set: {
      "value.total_time": { $add: ["$value.total_time", "new.value.total_time"] },
      "value.count": { $add: ["$value.count", "new.value.count"] } },
      "value.avg": { $divide: [ { $add: ["$value.count", "new.value.count"] }, { $add: ["$value.total_time", "new.value.total_time"] } ] }
    } },
    whenNotMatched: "insert"
  }}
])
</code></pre>
```

- ▶ Ejecutar una consulta sobre la nueva colección.

```
db.session_stats_agg.find().sort( { _id: 1 } )
```

Sugerencia: una vez finalizada la lectura, publica un mensaje en el foro de este tema dando tu punto de vista sobre las ventajas de usar Map-Reduce y Aggregate. Investiga un poco más sobre el tema y comenta si el hecho de que MongoDB use JavaScript dificulta de alguna manera el uso de Map-Reduce o Aggregate.

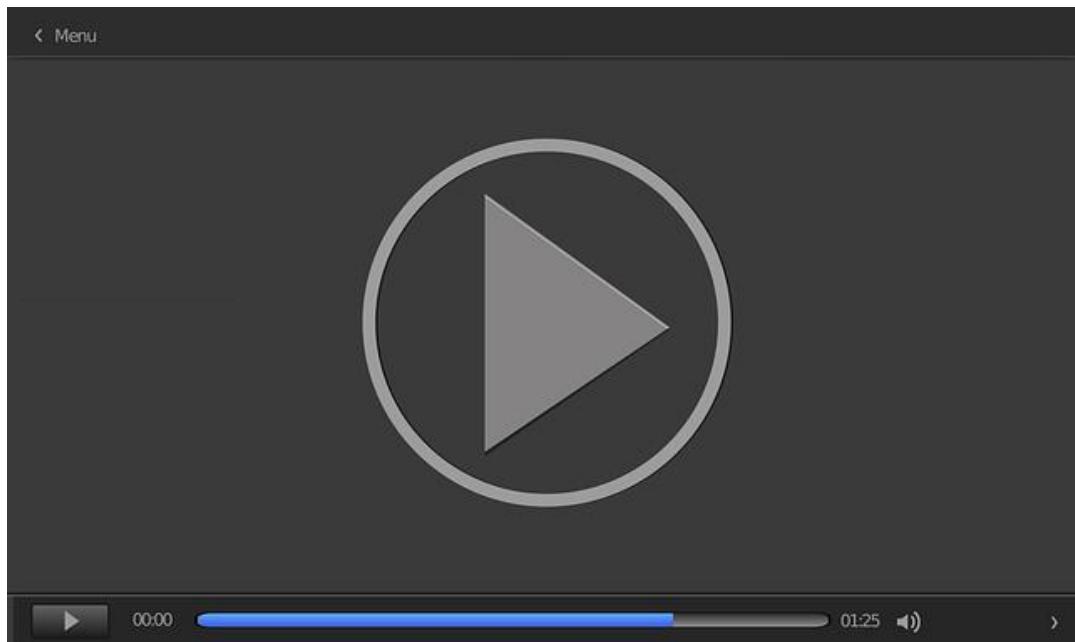
Después de leer todo el tema, el reto siguiente es que hagas un repaso sobre el uso de Map-Reduce y lo compares con Aggregate. En los vídeos siguientes el profesor repasará los casos prácticos y comentará algunos aspectos que se deben tener en cuenta a la hora de trabajar con estos mecanismos.



Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=fcd14622-e49b-4ed4-a223-aca7018b452a>

Vídeo 1. Map-Reduce vs. Aggregate, caso práctico 1.

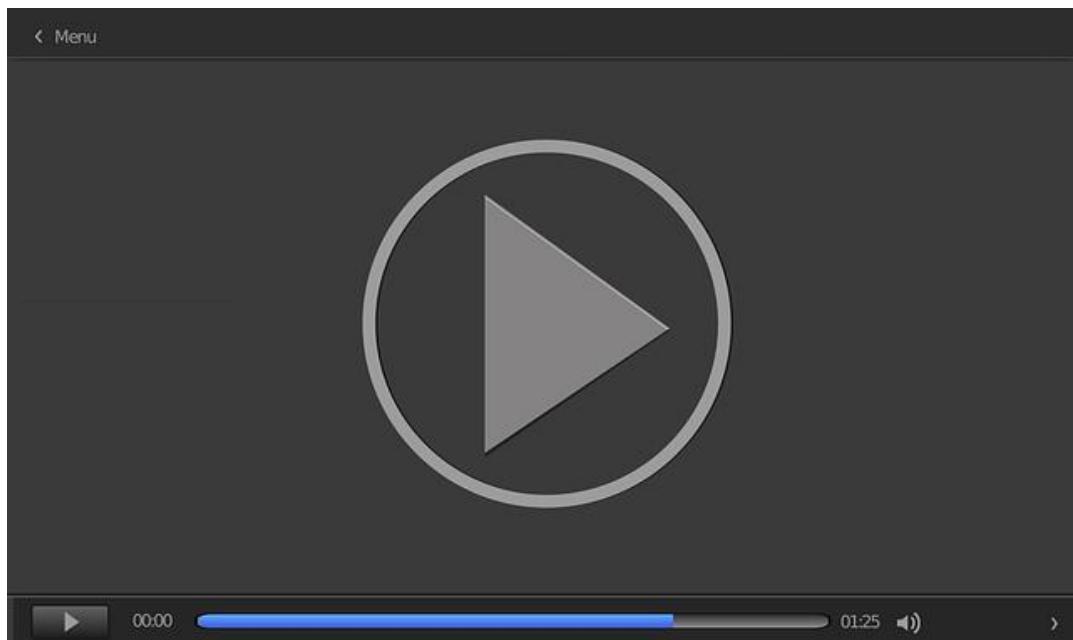


Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=11b76226-4a4b-40af-afc8-aca80003fd5c>

Vídeo 2. Map-Reduce vs. Aggregate, caso práctico 2.

En esta otra clase, «Utilización del *framework Aggregate*», se explica cómo utilizar el *framework Aggregate* con ejemplos prácticos.



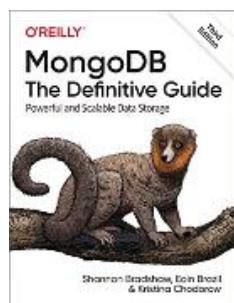
Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=b3110e8c-63e3-4621-b87a-adbe00e864f7>

Vídeo 3. Utilización del *framework Aggregate*.

MongoDB: the definitive guide

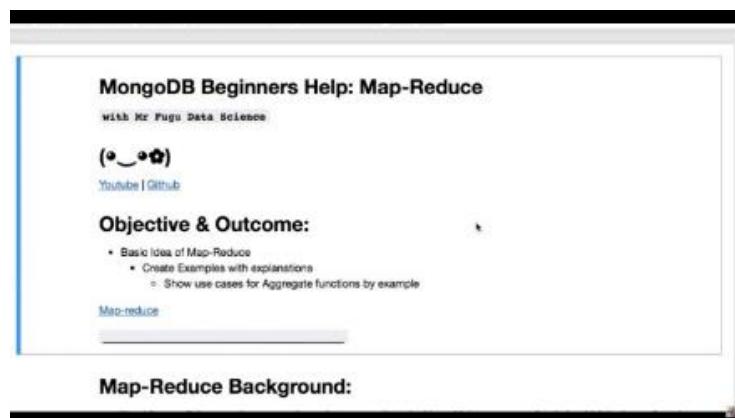
Chodorow, K., y Dirolf, M. (2019). *MongoDB: the definitive guide*. O'Reilly. <https://www.oreilly.com/library/view/mongodb-the-definitive/9781491954454/>



Este libro contiene información detallada sobre la creación, eliminación y modificación de documentos en MongoDB. El capítulo 6 detalla las funciones de agregación.

MongoDB Map-Reduce

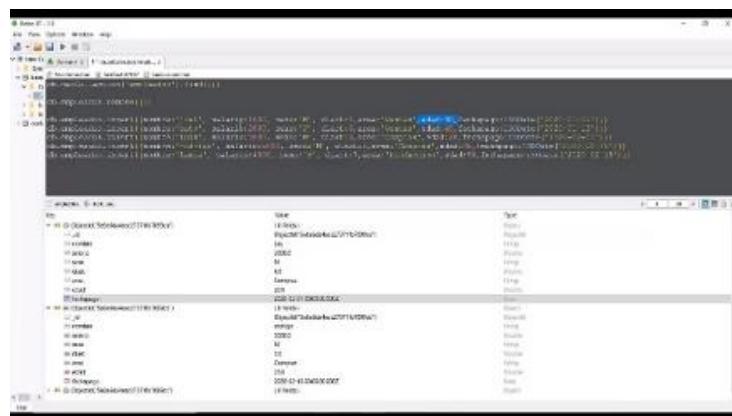
Sr. Fugu Data Science. (12 de mayo de 2020). *Tutorial de MONGODB: REDUCCIÓN DE MAPAS* [Archivo de vídeo]. <https://www.youtube.com/watch?v=MGZkKzfSqP4>



Esta es una guía para principiantes de **Map-Reduce (MongoDB)** con ejemplos y casos de uso. También habrá ejemplos de agregación, para mostrar cuándo es posible que Map-Reduce no encaje.

MongoDB Aggregation Framework

Herrera, J. [Programar es fácil]. (8 de abril de 2020). *Tutorial MongoDB 4.2 Parte 10 Agregaciones: Instrucción Aggregate \$match \$group \$sum \$avg \$multiply* [Archivo de vídeo]. <https://www.youtube.com/watch?v=OV2byyfKHIw>



Esta es una guía de **Aggregate** en **MongoDB** con ejemplos y casos de uso. También habrá ejemplos de agregación, para mostrar cuándo es posible que Map-Reduce no encaje.

Aggregation Framework

MongoDB. (25 de junio de 2014). *The Aggregation Framework* [Archivo de vídeo].

<https://www.mongodb.com/presentations/aggregation-framework-0>



La introducción a MongoDB presentada durante los primeros minutos de este vídeo incluye una breve descripción de las características del sistema. También presentan un ejemplo con las diferencias entre bases de datos relacionales y MongoDB.

Documentación oficial de MongoDB

MongoDB. (2021). Documentation [Página web]. <http://docs.mongodb.org/>

Dentro del sitio oficial con la documentación del sistema MongoDB, el contenido relevante para este tema es la sección MongoDB Map-Reduce y Aggregation Framework.



Bibliografía

Bradshaw, S., Brazil, E., y Chodorow, K. (2019). *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media.

Castillo, J. N., Garcés, J. R., Navas, M. P., Segovia, D. F. J., y Naranjo, J. E. A. (2017). Base de Datos NoSQL: MongoDB vs. Cassandra en operaciones CRUD (Create, Read, Update, Delete). *Revista Publicando*, 4(11 (1)), pp. 79-107.

Giamas, A. (2017). *Mastering MongoDB 3.x: an expert's guide to building fault-tolerant MongoDB applications*. Packt Publishing.

Tiwari, S. (2011). *Professional NoSQL* (pp. 97-135, 217-232). John Wiley & Sons.

1. ¿Para qué son útiles las funciones de agregación?

 - A. Para agrupar datos.
 - B. Para realizar cálculos.
 - C. Para crear nuevas colecciones.
 - D. Todas las anteriores son correctas.

2. ¿Qué método puede utilizarse en MongoDB para agregar información de documentos en una *collection*?

 - A. sum.
 - B. Aggregate, a partir de la versión 2.2.
 - C. Map-Reduce.
 - D. Las respuestas B y C son correctas.

3. ¿Cuál de las siguientes son operaciones específicas de agregación?

 - A. sum.
 - B. Map-Reduce.
 - C. count.
 - D. Las respuestas A y C son correctas.

4. ¿Cuál es el objetivo principal de la función *map*?

 - A. Generar los pares clave-valor.
 - B. Realizar operaciones con los atributos de la colección.
 - C. Operar sobre los pares clave-valor.
 - D. Crear una nueva colección.

5. ¿Cuál es el objetivo principal de la función *reduce*?
 - A. Generar los pares clave-valor.
 - B. Realizar operaciones con los atributos de la colección.
 - C. Operar sobre los pares clave-valor.
 - D. Crear una nueva colección.
6. ¿Qué comando se utiliza en una función *map* para generar el par clave-valor que será procesado posteriormente?
 - A. generate.
 - B. return.
 - C. emit.
 - D. yield.
7. ¿Cuál de las siguientes es una ventaja de *framework* de agregación de MongoDB?
 - A. Rendimiento.
 - B. Potencia.
 - C. Simplicidad.
 - D. Las respuestas A y C son correctas.
8. ¿En qué está modelado el *framework* de agregación?
 - A. Funciones.
 - B. Etapas.
 - C. Sentencias SQL.
 - D. Agrupaciones.

9. ¿Qué campo es obligatorio especificar en *framework* de agregación?

- A. _id.
- B. Object.
- C. \$sum.
- D. \$group.

10. ¿Cuál de los siguientes es un operador del *framework* de agregación?

- A. \$gt.
- B. \$map.
- C. \$year.
- D. Todos los anteriores son correctos.

Bases de Datos para el Big Data

Tema 5. Gestión de MongoDB

Índice

[Esquema](#)

[Ideas clave](#)

[5.1. Introducción y objetivos](#)

[5.2. Seguridad](#)

[5.3. Respaldo](#)

[5.4. Rendimiento](#)

[5.5. Sharding](#)

[A fondo](#)

[MongoDB Sharding 101](#)

[MongoDB: how to deploy a MongoDB replica set](#)

[Documentación oficial de MongoDB](#)

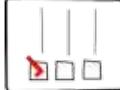
[The database for modern applications: MongoDB](#)

[A Cookbook for MongoDB](#)

[MongoDB: The Definitive Guide](#)

[Bibliografía](#)

[Test](#)

GESTIÓN DE MONGODB			
Seguridad	Respaldo	Rendimiento	Sharding
Autenticación y listas de acceso: permiten controlar permisos de interacción con bases de datos.	La herramienta Mongodump se utiliza para crear una copia de respaldo de una base de datos en MongoDB.	Los índices permiten ejecutar consultas de forma eficiente.	Sharding: método utilizado por MongoDB para distribuir datos en varios servidores y aumentar así su capacidad.
Para restringir conexiones desde una interfaz en específico se debe indicar la dirección IP en el parámetro <code>-Bind_ip</code> .	La herramienta Monitorestore permite restaurar una copia de respaldo en el sistema MongoDB.		Cada shard está encargado de almacenar la información. Cada shard es un replica set , por lo que brinda alta disponibilidad y consistencia en datos.
La seguridad del entorno MongoDB puede incrementarse con el uso de un firewall .	MongoDB brinda capacidades de replicación, para lograr reducirse e incrementar la capacidad.	 	Este aumento de rendimiento se ve reflejado en el número de documentos analizados y, por consiguiente, en el tiempo de respuesta.

5.1. Introducción y objetivos

Este tema se enfoca en operaciones y parámetros para la gestión de bases de datos en el sistema MongoDB. La mejor forma para estudiar este tema es mediante la lectura de las Ideas clave, sobre todo por el alto número de conceptos que aparecen relacionados con el rendimiento de las bases de datos. Además, la realización de las actividades permitirá poner en práctica la aplicación de estos conceptos.

Al contrario que con las operaciones básicas, verificar el resultado de operaciones de gestión puede tener un grado mayor de complejidad, por lo que se recomienda validar las soluciones.

Un correcto tratamiento del almacenamiento de información requiere conocimientos sobre la gestión del sistema de base de datos. Es por esta razón que, además de estudiar las operaciones básicas para el tratamiento de datos en MongoDB, también se hará un repaso de operaciones de gestión sobre la base de datos.

Las operaciones de gestión que se describen en este tema toman en cuenta la gestión de parámetros de seguridad, que es de gran importancia cuando los datos que se manipulan son de carácter personal o privado. Otra parte importante es la migración de una base de datos, lo cual incluye la realización de copias de respaldo y la restauración de la copia, así como la replicación de nodos para respaldo.

Además, es necesario configurar la base de datos para lograr un rendimiento óptimo, lo cual puede realizarse a través de la creación de índices. Finalmente, se abordarán temas de repartición de la carga entre diferentes nodos gracias al *sharding*.

Los objetivos que se persiguen con este tema son los siguientes:

- ▶ Conocer los elementos mínimos que una correcta gestión de MongoDB debe incorporar.

Ideas clave

- ▶ Entender los requerimientos mínimos de seguridad en un despliegue de MongoDB.
- ▶ Aprender los pasos para configurar los procesos de respaldo, mejora del rendimiento y *sharding* en la base de datos.

5.2. Seguridad

Uno de los aspectos que tomar en cuenta en entornos de producción es la protección de los datos mediante medidas de seguridad. MongoDB permite restringir el acceso a las bases de datos mediante la autenticación del usuario. Inicialmente se debe crear un usuario administrador, el cual tendrá privilegios para crear usuarios y acceder a todas las bases de datos en el sistema.

Un usuario puede tener privilegios en diferentes bases de datos. Si necesita que un usuario tenga privilegios en varias bases de datos, crea un solo usuario con roles que otorguen los privilegios de base de datos aplicables en lugar de crear el usuario varias veces en diferentes bases de datos.

El proceso para crear un administrador consiste en utilizar el comando `createUser` en la base de datos `admin`:

```
use admin
db.createUser(<Nombre de usuario>, <Contraseña de usuario>)
```

Por defecto, el servicio `mongod` no requiere autenticación para interactuar con la base de datos. Este comportamiento puede cambiarse mediante el parámetro `--auth` al ejecutar `mongod` desde la línea de comando.

Para autenticarse desde la consola de MongoDB se utiliza el comando `db.auth`:

```
db.auth(<Nombre de usuario>, <Contraseña de usuario>)
```

La creación de usuarios en una base de datos se realiza con el mismo comando, con la diferencia de realizarlo dentro de la base de datos apropiada y no en `admin`. Además, en la creación de un usuario de base de datos, el comando `db.auth` puede recibir un tercer parámetro de valor booleano que indica si el usuario solamente tiene privilegios de lectura.

Además de asegurar el acceso a la base de datos a través de métodos de autenticación, en ocasiones es conveniente restringir los mecanismos de acceso a nivel de red. Por defecto, MongoDB espera conexiones desde cualquier interfaz de red del ordenador en el que se encuentra. Para restringir conexiones desde una interfaz en específico se debe indicar la dirección IP en el parámetro `--bind_ip` al iniciar el servicio `mongod`.

Además de las opciones de seguridad que ofrece Mongo, es posible añadir muchas otras externas, como podría ser la utilización de un *firewall* para restringir las comunicaciones del exterior o de otros dominios.

A continuación, se describen mediante un ejemplo los pasos necesarios para realizar una correcta restricción de acceso:

- ▶ **Paso 1:** se crea el usuario administrador:

```
db.createUser({  
    user: "admin",  
    pwd: "mipassword",  
    roles:[  
        "clusterAdmin",  
        "readAnyDatabase",  
        "readWriteAnyDatabase",  
        "userAdminAnyDatabase",  
        "dbAdminAnyDatabase"  
    ]  
});
```

- ▶ **Paso 2:** parar el servidor y volver a iniciar con la autenticación activada. Este proceso se puede realizar de dos maneras:

- Arrancamos el servidor con el siguiente comando:

```
mongod --auth --port 27017 --dbpath /data/db1
```

- ▶ Añadimos la siguiente línea al archivo de configuración de Mongo (*/bin/mongod.cfg*):

```
security:  
authorization: enabled
```

- ▶ **Paso 3:** se crea el usuario con los roles establecidos.

- Se accede como administrador:

```
mongo -u "usuario" -p "pass"
```

- ▶ Se accede a la base de datos sobre la que se quiere crear el usuario:

```
use DatabaseForSensors
```

- ▶ Se crea el usuario para esa base de datos:

```
db.createUser({  
  user: "luis",  
  pwd: "jf9paus90D",  
  roles: [ { role: "readWrite", db: "DatabaseForSensors" } ]  
})
```

MongoDB ofrece funciones integradas que proporcionan los diferentes niveles de acceso comúnmente necesarios en un sistema de base de datos. Existen roles de usuario de base de datos incorporados y funciones de administración de bases de datos en cada base de datos. La base de datos admin contiene funciones adicionales.

Cada base de datos incluye los siguientes roles:

- ▶ **read:** proporciona la capacidad de leer datos de todas las colecciones que no son del sistema y de las siguientes colecciones del sistema: system.indexes, system.js y system.namespaces .

- ▶ **readWrite**: proporciona todos los privilegios del rol de lectura y la capacidad de modificar datos de todas las colecciones que no son del sistema y de la colección system.js ..

Cada base de datos incluye los siguientes roles de administración de base de datos:

- ▶ **dbAdmin**: proporciona la capacidad de realizar tareas administrativas, tales como tareas relacionadas con el esquema, indexación y recopilación de estadísticas. Esta función no otorga privilegios para la administración de usuarios y roles.
- ▶ **dbOwner**: proporciona la capacidad de realizar cualquier acción administrativa en la base de datos. Este rol combina los privilegios otorgados por las funciones `readWrite`, `dbAdmin` y `userAdmin`.
- ▶ **userAdmin**: proporciona la capacidad de crear y modificar roles y usuarios en la base de datos actual. Dado que el rol `userAdmin` permite a los usuarios otorgar cualquier privilegio a cualquier usuario, incluyendo a ellos mismos, el rol también indirectamente proporciona acceso de *superusuario* a la base de datos o, si se extiende a la base de datos de administración, al clúster.

La base de datos `admin` incluye las siguientes funciones para administrar todo el sistema en lugar de una base de datos específica:

- ▶ **clusterAdmin**: proporciona el mayor acceso de administración de clústeres. Este rol combina los privilegios otorgados por las funciones `clusterManager`, `clusterMonitor` y `hostManager`. Además, la función proporciona la acción `dropDatabase`.
- ▶ **clusterManager**: proporciona acciones de administración y supervisión en el clúster. Un usuario con esta función puede acceder a las bases de datos de configuración y local, que se utilizan en *sharding* y replicación, respectivamente.

Ideas clave

- ▶ **clusterMonitor**: proporciona acceso de solo lectura a las herramientas de supervisión, como MongoDB Cloud Manager y el agente de supervisión Ops Manager.
- ▶ **hostManager**: proporciona la capacidad de supervisar y administrar servidores.

La base de datos `admin` incluye las siguientes funciones para realizar copias de seguridad y restaurar datos:

- ▶ **backup**: proporciona los privilegios necesarios para realizar copias de seguridad de los datos. Esta función proporciona suficientes privilegios para utilizar el agente de copia de seguridad de MongoDB Cloud Manager, el agente de copia de seguridad de Ops Manager o para utilizar `mongodump`.
- ▶ **restore**: proporciona los privilegios necesarios para restaurar los datos con `mongorestore` sin la opción `--oplogReplay` o sin los datos de la colección `system.profile`

Estas funciones en la base de datos de administración se aplican a todas las bases de datos local y `config` excepto en una instancia `mongod`:

- ▶ **readAnyDatabase**: proporciona los mismos permisos de solo lectura, excepto los que se aplican sobre todas las bases de datos, las locales y el `config` del clúster. El rol también proporciona la acción `listDatabases` en el clúster como un todo.
- ▶ **readWriteAnyDatabase**: proporciona los mismos permisos de lectura y escritura que `readWrite`, excepto que se aplica a todas las bases de datos local y `config` del clúster. El rol también proporciona la acción `listDatabases` en el clúster como un todo.
- ▶ **userAdminAnyDatabase**: proporciona el mismo acceso a las operaciones de administración de usuarios como `userAdmin`, excepto que se aplica a todas las bases de datos local y `config` del clúster.

- ▶ **dbAdminAnyDatabase**: proporciona el mismo acceso a las operaciones de administración de bases de datos que `dbAdmin`, excepto que se aplica a todas las bases de datos locales y de configuración del clúster. El rol también proporciona la acción `listDatabases` en el clúster como un todo.

MongoDB proporciona una serie de funciones integradas. Sin embargo, si estos roles no pueden describir el conjunto de privilegios deseado, se puede crear nuevos roles.

Para añadir una función, MongoDB proporciona el método `db.createRole()`. MongoDB también proporciona métodos para actualizar las funciones existentes definidas por el usuario.

A partir de MongoDB Enterprise 3.2, es posible cifrar datos en la capa de almacenamiento con el cifrado en reposo nativo del motor de almacenamiento `WiredTiger`.

A partir de la versión 4.0, MongoDB deshabilita la compatibilidad con el cifrado TLS 1.0 en sistemas donde TLS 1.1+ está disponible. Consulta la documentación para más información.

Se recomienda ejecutar procesos de MongoDB con una cuenta de usuario de sistema operativo dedicada. Lo correcto es que dicha cuenta tenga los permisos para acceder a los datos (sin permisos innecesarios).

5.3. Respaldo

La migración de base de datos de un servidor a otro suele ser necesaria, ya sea por una actualización o cambio de *hardware* o por motivos administrativos. Al gestionar un sistema MongoDB cuentas con dos opciones para realizar esta migración.

La primera consiste en copiar el directorio del sistema de ficheros a la nueva ubicación. Por defecto, los datos de MongoDB se almacenan en la ruta `/data/db` en Linux, o `C:\data\db` en Windows (ruta recomendada). Es importante tener en cuenta que **los datos no se copiarán adecuadamente si el servidor de MongoDB se está ejecutando en el momento de realizar la copia**.

Además, MongoDB permite exportar e importar bases de datos mediante dos herramientas a utilizar desde la línea de comandos: `mongodump` y `mongorestore`, respectivamente.

Mongodump hace una copia de seguridad de los datos conectándose a una instancia `mongod` o `mongos` en ejecución. Esta utilidad permite crear una copia de seguridad para un servidor, una base de datos o una colección completa, o puede utilizar una consulta para copiar solo parte de una colección.

Se puede ejecutar `mongodump` sin argumentos. Esto creará una copia de seguridad de la base de datos en la carpeta `dump/` siempre y cuando la base de datos se encuentre en el sistema local (`localhost` o dirección IP `127.0.0.1`) y en el puerto **27017**.

El formato de datos que utiliza `mongodump` a partir de la versión 2.2 o posterior es incompatible con las versiones anteriores, por lo que no se deben utilizar versiones recientes de este comando para realizar copias de seguridad de datos antiguos.

A continuación, se puede ver el comando completo que se debería ejecutar para hacer la copia de seguridad especificando equipo, puerto y directorio de salida:

```
mongodump --host mongoHost.es --port 27017 --out /data/backup/
```

Si se quiere especificar una base de datos y una colección se deben utilizar las opciones `--collection` y `--db` de la siguiente forma:

```
mongodump --collection miColeccion --db miBBDD
```

Gracias a las opciones `--username user` `--password pass` se podrán especificar el nombre de usuario y la clave de acceso de la base de datos en el caso que fuera obligatorio.

También se puede utilizar la opción `--oplog` para recopilar las entradas de *oplog* para crear una instantánea puntual de una base de datos dentro de un conjunto de réplicas. Con `--oplog`, mongodump copia todos los datos de la base de datos origen, así como todas las entradas *oplog* desde el principio hasta el final del proceso de copia de seguridad.

Esta operación, junto con `mongorestore --oplogReplay`, permite restaurar una copia de seguridad exactamente igual al momento que se completó el volcado con `mongodump`.

La utilidad `mongorestore` restaura una copia de seguridad creada por `mongodump`. Por defecto, `mongorestore` busca la copia de seguridad de la base de datos en el directorio `dump/`. El proceso restaura los datos conectándose directamente a un `mongod` o `mongos`. `Mongorestore` puede restaurar una copia de seguridad completa de la base de datos o de un subconjunto de la copia de seguridad. Un ejemplo es el que se puede ver a continuación:

```
mongorestore --port 27017 /db/backup/mongodump-2013-10-24
```

Donde 27017 es el puerto de conexión *localhost* y /db/backup/mongodump-2013-10-24 el archivo y directorio donde se encuentra la copia creada anteriormente.

Nota: a partir de MongoDB 3.6, los binarios de MongoDB, mongod y mongos , se vinculan a *localhost* de forma predeterminada. Desde las versiones 2.6 a 3.4, solo los binarios de los paquetes oficiales de MongoDB RPM (Red Hat, CentOS, Fedora Linux y derivados) y DEB (Debian, Ubuntu y derivados) se vincularían a *localhost* de forma predeterminada.

Un ejemplo más complejo en el que se pueden ver todas las opciones del comando mongorestore es el que se puede ver a continuación:

```
mongorestore --host mongodb1.example.net --port 3017 --username user --password pass /db/backup/mongodump-2013-10-24
```

En este ejemplo, además, se ha especificado el *host*, el nombre de usuario y la contraseña de acceso de la base de datos donde se va a realizar la restauración.

Una vez comprendido cómo se puede hacer una copia de seguridad de la base de datos y cómo se puede restaurar la copia, es muy importante saber qué mongo proporciona otro mecanismo para mantener seguros los datos mediante la replicación de datos en varios nodos.

Cuando se realiza una copia de seguridad, se está haciendo una foto en un instante de tiempo dado; en cambio, con las réplicas tenemos la información duplicada en varios nodos en todo momento.

Nota: las utilidades mongodump y mongorestore funcionan con volcados de datos BSON y son útiles para crear copias de seguridad de pequeñas implementaciones. Para copias de seguridad resistentes y no disruptivas, usa un sistema de archivos o una función de instantánea de disco a nivel de bloque, como los métodos descritos en el documento «Métodos de copia de seguridad de MongoDB» (documentación oficial).

Otros métodos de *backup* en MongoDB

Cuando se implementa MongoDB en producción, las copias de seguridad se deben gestionar de una forma más profesional y robusta. Para ello, es importante que conozcas qué alternativas existen en entornos productivos, según la instalación que se tenga de MongoDB.

- ▶ **BackUp con Atlas:** es la opción de servicio de MongoDB alojado en la nube, la cual ofrece dos métodos totalmente administrados para realizar copias de seguridad:
 - **Cloud BackUp:** utiliza la funcionalidad de *snapshot* nativa del proveedor de servicios en la nube de la implementación para ofrecer opciones de copia de seguridad sólidas.
 - Suelen disponer de *on-demand snapshots* (permiten activar una instantánea inmediata en un momento dado).
 - **BackUp continuas en el Cloud:** permiten programar copias de seguridad periódicas para su implementación.
 - **Legacy Backups:** o copias de seguridad heredadas, las cuales realizan copias de seguridad incrementales de los datos en su implementación (esto actualmente está en desuso).
- ▶ **BackUp con Cloud Manager:** con esta alternativa se respalda continuamente los conjuntos de réplicas de MongoDB y los clústeres fragmentados leyendo los datos de registro de operaciones de su implementación. Este servicio crea *snapshots* de sus datos a intervalos establecidos y también puede ofrecer recuperación en un momento determinado de conjuntos de réplicas y clústeres fragmentados previamente.

Ideas clave

- ▶ **Ops Manager:** con esta opción, los suscriptores de MongoDB pueden instalar y ejecutar el mismo software central que impulsa MongoDB Cloud Manager en su propia infraestructura. Ops Manager es una solución local que tiene una funcionalidad similar a MongoDB Cloud Manager y está disponible con suscripciones Enterprise Advanced.
- ▶ **BackUp con copia de datos subyacentes:** permite crear o bien un respaldo de una implementación de MongoDB haciendo una copia de los archivos de datos subyacentes de MongoDB, o bien puedes copiar los archivos directamente usando *cp*, *rsync* o una herramienta similar si la implementación no soporta instantáneas.

La replicación es una característica de MongoDB que permite la redundancia de datos, así como incrementar su disponibilidad. La arquitectura básica de replicación en MongoDB sigue un **modelo Maestro-Esclavo**. La instancia maestra recibe operaciones de escritura y estas son aplicadas en la instancia dependiente para mantener la consistencia de los repositorios de datos.



Figura 1. Representación de nodos Maestro-Esclavo para MongoDB.

La instancia maestra se ejecuta mediante la inclusión del parámetro `--master` en la ejecución del servidor de MongoDB. La instancia secundaria se ejecuta mediante la inclusión del parámetro `--slave`, siguiendo ambos la siguiente sintaxis:

Ejecución de instancia maestra:

```
mongod --master
```

Ejecución de un nodo secundario/esclavo:

```
mongod --slave --source <dirección de nodo maestro>
```

En MongoDB se ha hecho un refinamiento sobre el modelo Maestro-Esclavo y se le ha definido como conjunto de réplicas o *replica set*. Este método permite una recuperación a fallos de forma automática, y es la forma recomendada de implementar replicación de datos en MongoDB.

Una de las principales características de los *replica set* es que no definen un nodo maestro estáticamente, sino que es asignado al azar y esta asignación se actualiza dependiendo de la disponibilidad del nodo. Los nodos mantienen control de qué otros nodos son accesibles mediante el envío de *pings* cada dos segundos.

La configuración mínima recomendada para una *replica set* consiste en tres nodos: dos nodos serán instancias de `mongod` con acceso a los datos; uno de estos nodos será el nodo primario mientras que el otro replicará las operaciones que se realicen sobre el primario.

El tercer nodo, denominado árbitro, es utilizado solamente para decidir qué nodo debe ser asignado como primario. La siguiente figura muestra el diagrama de la arquitectura descrita.

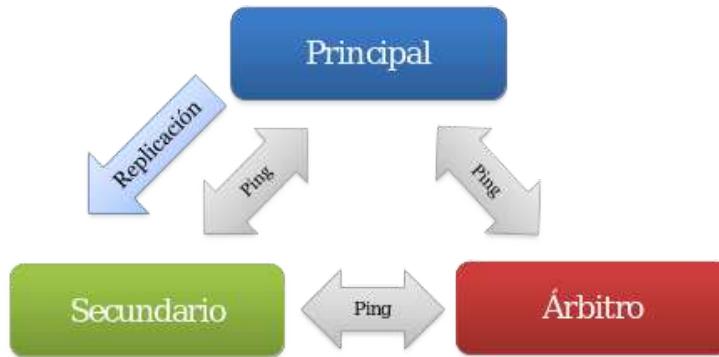


Figura 2. Ejemplo de la replicación con nodo árbitro en MongoDB.

Para iniciar una instancia de MongoDB como parte de un *replica set*, el comando sigue el siguiente formato:

```
mongod --replSet <nombre de conjunto de réplicas>/<dirección de otro host>
```

Después de iniciar todos los nodos, es necesario inicializar el conjunto conectándose a la base de datos admin de alguno de ellos y ejecutando el comando `replSetInitiate` con la siguiente sintaxis:

```
db.runCommand({“replSetInitiate”: {  
    _id: <Identificador de conjunto>,  
    members: <Listado de nodos en el Replica Set>  
}})
```

El listado de nodos en el *replica set* es un array de objetos con la información de cada nodo. Entre los atributos de cada uno de estos objetos se puede enfatizar los siguientes:

- ▶ **_id**: identificador secuencial obligatorio, el primer nodo en el *replica set* tiene un identificador igual a 0.
- ▶ **host**: cadena de texto obligatoria con el nombre del *host* de este nodo.
- ▶ **arbiterOnly**: booleano para indicar si el nodo es solamente árbitro.

- ▶ **priority**: valor de 0 a 1000 que indica la prioridad de dicho nodo para ser elegido primario.
- ▶ **votes**: indica el número de votos de este nodo para la elección de un nuevo nodo primario. Se sugiere indicar un solo voto por nodo y, en caso de cambiar esta asignación, simular detenidamente los efectos de este cambio.
- ▶ **hidden**: booleano que indica si el nodo debe ocultarse del listado dado por el comando `db.isMaster()` .
- ▶ **buildIndexes**: booleano que indica si este nodo creará índices. Por defecto es verdadero y solo será falso en aquellos casos donde se sabe que el nodo no será primario, es decir, tiene un valor 0 en el atributo `priority` .
- ▶ **slaveDelay**: número de segundos de retraso que tendrá un nodo secundario. Debe ser mayor que cero solo para aquellos nodos que no serán primarios nunca.
- ▶ **tags**: documento JSON con pares clave-valor definidos por el administrador de la base de datos.

5.4. Rendimiento

El incremento de rendimiento en el sistema MongoDB puede realizarse a través de acciones como la creación de índices, los cuales mejoran el tiempo de ejecución de consultas sobre *collections*.

Es recomendable analizar los atributos de un documento sobre los cuales será beneficioso aplicar un índice. Por ejemplo, si es común filtrar documentos en la *collection* usuarios por el atributo `email`, será conveniente crear un índice sobre dicho atributo.

El comando brindado para crear un índice es `createIndex` y posee el siguiente formato:

```
db.<nombre de collection>.createIndex(<claves>, <opciones>)
```

Ambos argumentos en la llamada a la función son objetos en formato JSON. El objeto `claves` indica los atributos sobre los cuales se creará el índice, teniendo como valor el número 1 si el índice se creará de forma ascendente y -1 si es de forma descendente.

- ▶ El objeto `opciones` puede contener los siguientes atributos, todos opcionales:
 - ▶ **background**: valor booleano para indicar si el índice debe crearse sin bloquear otras actividades en la base de datos. Por defecto es `false`.
 - ▶ **unique**: valor booleano para indicar si un atributo no acepta valores repetidos. Por defecto es `false`.
 - ▶ **name**: cadena de texto para especificar el nombre del índice.

- ▶ **dropDups**: valor booleano que indica la creación de un índice único y la eliminación de documentos repetidos en la *collection*. Por defecto es `false`.
- ▶ **sparse**: valor booleano para indicar que el índice se aplica solamente sobre los documentos con los atributos indicados en el campo de claves.
- ▶ **v**: indica el número de versión del índice a crear. Antes de la versión 2.0 de MongoDB este valor es 0, para versiones posteriores el valor es 1.

Un índice puede eliminarse utilizando el comando `dropIndex`, siguiendo la sintaxis apropiada:

```
db.<nombre de collection>.dropIndex(<índice>)
```

El argumento índice puede ser una cadena de texto con el nombre del índice a eliminar o un objeto con el mismo contenido que el utilizado en el argumento de claves en `createIndex`.

Hay dos métricas que permiten medir el rendimiento:

- ▶ El tiempo de búsqueda: cuando se utilicen índices se podrá apreciar que el tiempo de búsqueda se ve reducido. Este efecto es más notable cuando el número de documentos es elevado.
- ▶ El número de documentos analizados en la búsqueda. La optimización de los índices se debe a que el número de documentos en los que el motor de la base de datos realiza la búsqueda es menor, llegando en las consultas exactas a ser el mismo número de documentos que devuelve la búsqueda.

Para esta métrica no influye el número de documentos que tenga la colección, puesto que, aunque la colección tenga pocos registros, se podrá ver en esta métrica que es inferior al total de ellos.

Se supone que se tiene el siguiente esquema de documento de la colección `users`:

```
{  
    userId: "Fer",  
    name: "Fernando Alonso",  
    age: 36,  
    hobbies: ["motor", "bike"],  
    info:{  
        friend: "Pedro"  
    },  
    localtion: [34,45]  
}
```

Algunos ejemplos de indexación pueden ser los siguientes:

- ▶ **Simple:** insertamos un índice en orden ascendente para el atributo `name`. El comando para insertar el índice y la búsqueda que lo aplica son los siguientes:

```
db.users.createIndex( { "name" : 1} )  
db.usessr.find({ name: "Fernando Alonso" })
```

- ▶ **Compuesto:** se pueden usar para consultar uno o varios campos sin incluir todos. El siguiente ejemplo permitirá optimizar la búsqueda para `name`, o para `name` y `age`. Si se quiere hacer solo sobre `age`, habría que crear un índice simple para este atributo. En el siguiente ejemplo se crea un índice para `name` y `age` y se crea la consulta que se aprovecha de esta optimización:

```
db.users.createIndex( { "name" : 1, "age": -1 } )  
db.usessr.find( {name: "Fernando Alonso", age:36 } )
```

- ▶ **Subdocumento:** si el atributo pertenece a un objeto, es necesario llegar al nivel deseado para asegurar el índice. El insertado del índice y la consulta que utiliza la optimización son las siguientes:

```
db.users.createIndex( { "info.friend" : 1 } )  
db.user.find({ info.friend: "Pedro"})
```

- ▶ **Arrays:** se pueden crear índices sobre claves que contienen `arrays`; para ello se

crea un valor en el índice para cada valor del *array*. Se denominan índices *multikey* y no hace falta intimidarlo explícitamente. El insertado del índice y la consulta que utiliza la optimización son las siguientes:

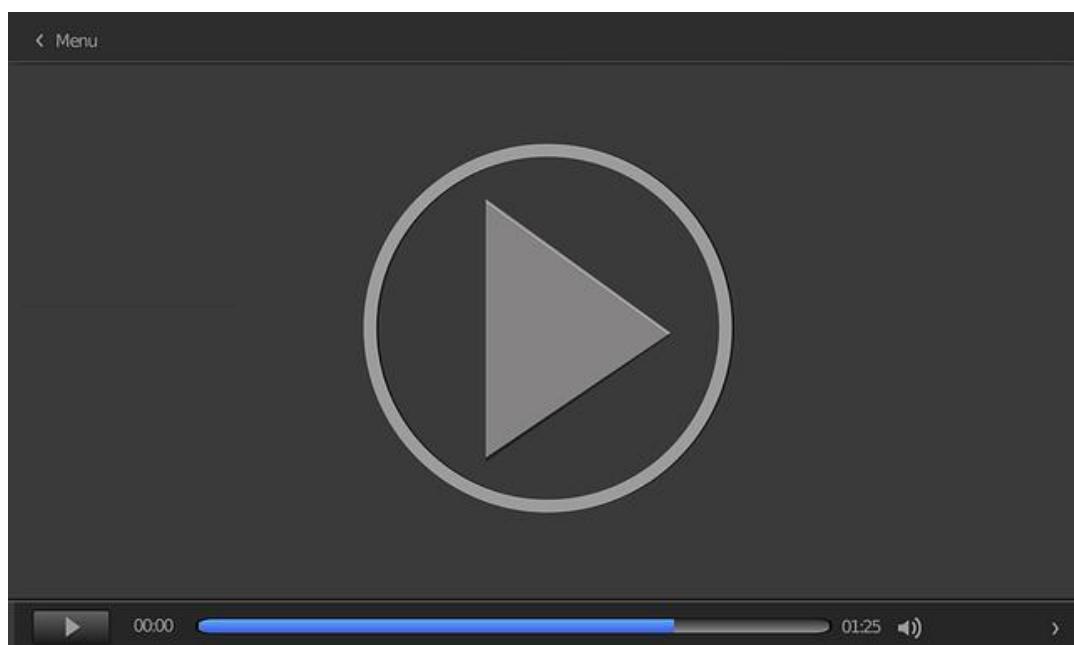
```
db.users.createIndex( { "hobbies": 1} )  
db.users.find({ hobbies: "motor" })
```

- ▶ **2d**: los documentos deben tener una clave que contenga un *array* de dos coordenadas. Estos índices son útiles cuando permiten utilizar operadores especiales como `$near`. El insertado del índice y la consulta que utiliza la optimización son las siguientes:

```
db.users.createIndex( { "location": 2d} )  
db.users.find({ location: {$near: [25,32] } }).limit(6)
```

Nota: a partir de la versión 3.0 de MongoDB, `createIndex` fue el método que reemplazó a `ensureIndex`.

Para comprender el uso de índices en MongoDB, revisa esta lección titulada «Creación de índices y análisis de consultas». En ella se muestran algunos ejemplos de índices en una base de datos, así como el análisis de rendimiento de una serie de consultas, antes y después de la creación de índices.



Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=abf6ef93-701b-4acc-807c-adbe00e854fb>

Vídeo 1. Creación de índices y análisis de consultas.

5.5. Sharding

Otro mecanismo para incrementar el rendimiento de una base de datos es escalar, ya sea de forma vertical u horizontal. La escalabilidad vertical consiste en incrementar o mejorar *hardware* (CPU y memoria RAM) que utiliza el sistema.

En contraste, la escalabilidad horizontal o *sharding* consiste en dividir el conjunto de datos y distribuirlo en varios servidores o *shards*. Así, el número de operaciones y la cantidad de datos se reparte entre el conjunto de *shards*. Si la base de datos crece, es suficiente con añadir un nuevo *shard* para poder almacenar más datos en un nuevo nodo.

Su principal utilidad es aumentar el rendimiento de las consultas distribuyendo la información entre varias máquinas y permitir añadir nuevas máquinas para poder aumentar la capacidad de almacenaje de información.

MongoDB brinda *sharding* a través de la definición de *sharded clusters*, los cuales están compuestos por tres tipos de componentes:

- ▶ **Shards:** están a cargo de almacenar la información. Cada *shard* es un *replica set*, por lo que brinda alta disponibilidad y consistencia de datos.
- ▶ **Query Routers:** son instancias del servicio `mongo` y sirven de interfaz entre los clientes y los *shards*.
- ▶ **Config Servers:** almacenan metadatos sobre la información almacenada en cada *shard*. Son consultados y modificados por los *query routers* para saber qué *shards* contienen información en específico. Los *sharded clusters* en un entorno de producción tienen exactamente tres *config servers*.

El primer paso para configurar un *sharded cluster* es iniciar los *config servers* a través del siguiente comando:

```
mongod --configsvr --dbpath <ruta a ficheros de datos> --port <puerto>
```

El puerto por defecto de un *config server* es el 27019. A continuación, se deben iniciar los *query routers* o instancias mongos :

```
mongos --configdb <nombres de los Config Servers, separados por comas>
```

El siguiente paso es añadir *shards* al clúster, primero conectándose al servicio mongos :

```
mongo --host <nombre de servidor ejecutando mongos> --port <puerto de mongos>
```

Y, después, utilizar el comando sh.addShard :

```
sh.addShard("<Identificador de Replica Set>/<Nombre de miembro del Replica Set>")
```

Finalmente, se debe habilitar la característica de *sharding* sobre la base de datos y sobre las *collections* a dividir:

```
sh.enableSharding("<Nombre de base de datos>")
sh.shardCollection("<Base de datos>.<Nombre de collection>", <Clave de Sharding>)
```

El segundo argumento indica el patrón que se utilizará para distribuir los datos en los

shards disponibles. Este argumento es un documento JSON con pares clave-valor para indicar los atributos que se utilizarán en la distribución. Es necesario que exista un índice sobre los atributos que se utilizan en esta configuración.

Map-Reduce y Aggregate admiten operaciones sobre colecciones en sistemas con *sharding*, tanto como entrada como como salida.

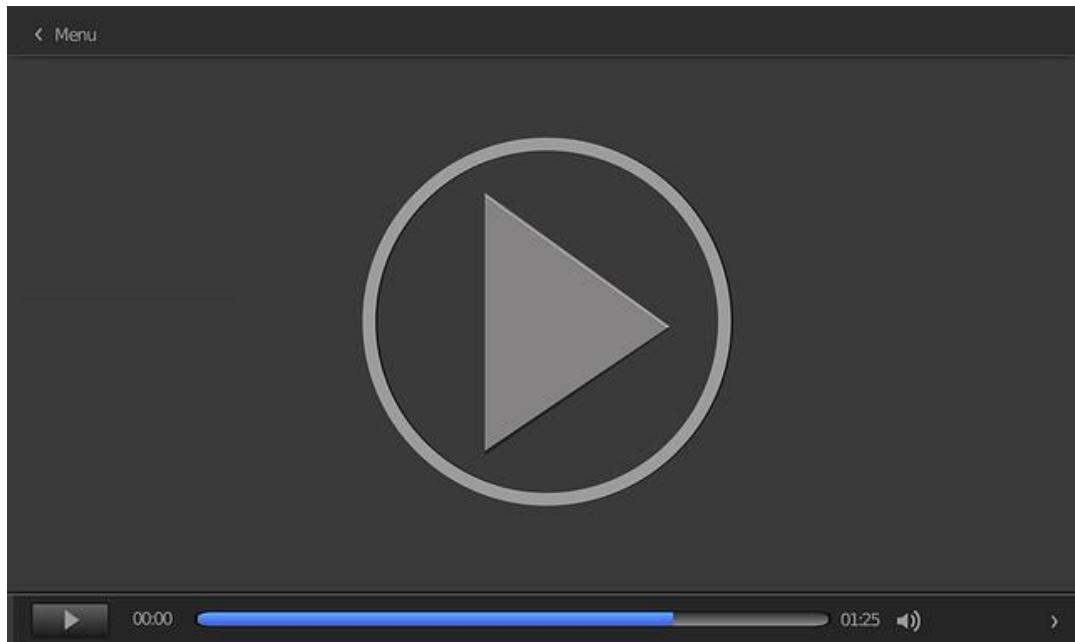
Cuando se utiliza la colección fragmentada como entrada, mongos enviará automáticamente el trabajo de agregación a cada fragmento en paralelo. No se requiere ninguna opción especial. Los mongos esperarán a que terminen los trabajos en todos los fragmentos.

Si el campo `out` de una función de agregación tiene el valor `sharded`, MongoDB fragmenta la colección de salida usando el campo `_id` como la clave `shard`.

Si la colección de salida no existe, MongoDB crea y fragmenta la colección en el campo `_id`. Para una colección nueva o vacía, MongoDB utiliza los resultados de la primera etapa de la operación de agregación para crear los trozos iniciales distribuidos entre los fragmentos. Mongos despacha en paralelo una tarea de postprocesamiento a cada fragmento que posee un pedazo.

Durante el postprocesamiento, cada fragmento extraerá los resultados de sus propios trozos, para finalmente ejecutar una reducción final y escribir el resultado localmente a la colección de salida.

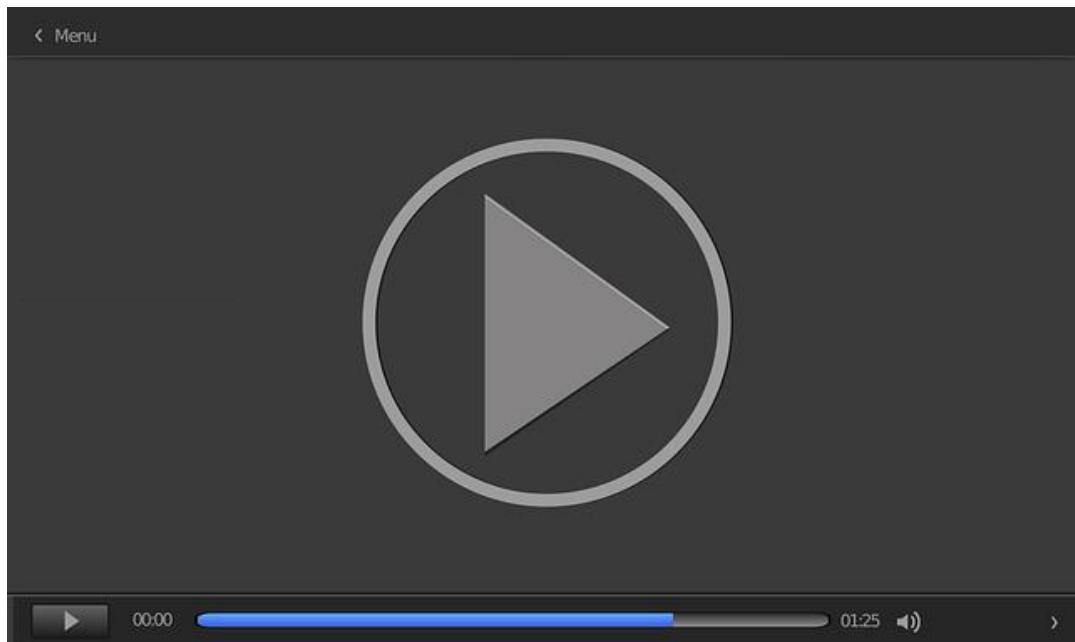
Después de leer todo el tema, lo siguiente es revisar los videos sobre MongoDB que se presentan a continuación. En ellos se describen de forma resumida los aspectos más relevantes en la gestión de MongoDB y algunas recomendaciones de cara a la administración de usuarios de la base de datos y al uso de clústeres.



Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=12f72116-7ef2-4774-b5d4-aca800dccc99>

Vídeo 2. Aspectos relevantes en la gestión de MongoDB: seguridad.



Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=e63d0f31-f1be-4963-b92f-aca800e25f47>

Vídeo 3. Aspectos relevantes en la gestión de MongoDB: exportar/restaurar y uso de índices.

MongoDB Sharding 101

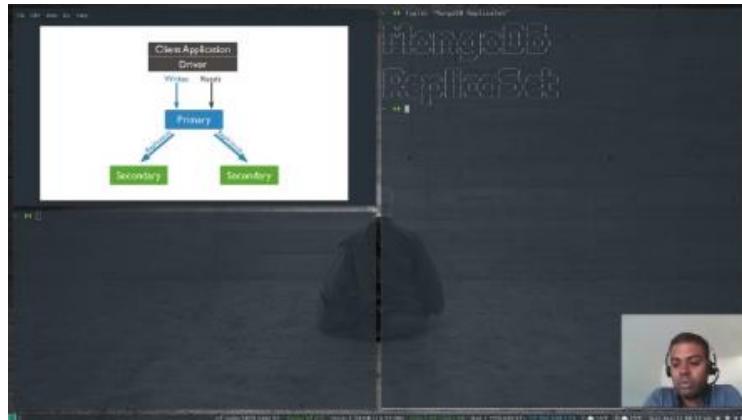
Percona. (4 de septiembre de 2018). *MongoDB Sharding 101 - MongoDB Sharding Tutorial* [Archivo de vídeo]. <https://www.youtube.com/watch?v=F7kyNeAPym4>



Este tutorial es una continuación de temas avanzados para el DBA. En él se comparten las mejores prácticas y consejos sobre cómo realizar las actividades más habituales. Se usa comúnmente para desarrollo o pruebas. El tutorial abarca los sistemas integrados, conjunto de réplicas, escalamiento horizontal, capacidad para elegir un nuevo primario en caso de falla (los datos son los mismos en las réplicas), réplica asincrónica, maestro único-primario.

MongoDB: how to deploy a MongoDB replica set

Just me and Opensource. (12 de agosto de 2019). [MongoDB 6] How to Deploy a MongoDB ReplicaSet [Archivo de vídeo]. <https://www.youtube.com/watch?v=Q2IJH156SUQ>



En este vídeo, se habla sobre el conjunto de réplicas MongoDB y cómo implementarlo. También repasa la simulación de una falla y demuestra cómo ocurren las elecciones primarias. Los conjuntos de réplicas se utilizan para escenarios de alta disponibilidad y recuperación ante desastres.

Documentación oficial de MongoDB

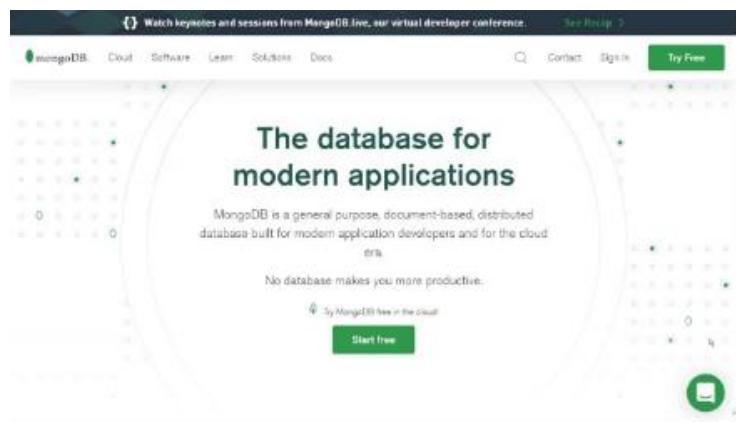
MongoDB. (2021). Documentation [Página web]. <http://docs.mongodb.org/>

Dentro del sitio oficial con la documentación del sistema MongoDB, el contenido relevante para este tema es la sección MongoDB Replicaset y Sharding.



The database for modern applications: MongoDB

Be A Better Dev. (15 de julio de 2020). *The ULTIMATE MongoDB Tutorial for 2021* [Archivo de vídeo]. https://www.youtube.com/watch?v=bo_M_BDcCbg



El video es un tutorial de MongoDB explicado por miembros del equipo de desarrollo de MongoDB. En media hora describen los aspectos más relevantes de la base de datos, así como la forma de optimizar las consultas más comunes. Dentro de estos aspectos se mencionan cuestiones relevantes con base de datos distribuida.

A Cookbook for MongoDB

Recurso disponible en la web oficial de libro, última visita el 01/12/2020.

Este sitio web contiene una serie de recetas de soluciones a escenarios comunes en el desarrollo de aplicaciones. Por ejemplo, una de las recetas indica cómo contar los tags asociados a posts en un sitio web, utilizando el método Map-Reduce.

Accede al recurso a través del aula virtual o desde la siguiente dirección web: <http://cookbook.mongodb.org/>

MongoDB: The Definitive Guide

Chodorow, K. & Dirolf, M. (2019). MongoDB: The Definitive Guide. California: O'Reilly.

Este libro contiene información detallada sobre la creación, eliminación y modificación de documentos en MongoDB. El capítulo 4 se centra en la realización de consultas, mientras que el capítulo 6 detalla las funciones de agregación.

Accede al libro a través del aula virtual o desde la siguiente dirección
w e b : <https://www.oreilly.com/library/view/mongodb-the-definitive/9781491954454/>

Bibliografía

Ajdari, J., y Kasami, B. (2018). MapReduce Performance in MongoDB Sharded Collections. *International Journal Of Advanced Computer Science And Applications*, 9(6), pp. 115-120.

Bradshaw, S., Brazil, E., y Chodorow, K. (2019). *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media.

Dipina Damodaran, B., Salim, S., y Vargese, S. M. (2016). Performance evaluation of MySQL and MongoDB databases. *International Journal on Cybernetics & Informatics (IJCI)*, 5.

Membrey, P., Plugge, E., y Hawkins, D. (2011). *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. Apress.

1. ¿Qué comando de MongoDB permite crear una copia de respaldo de una base de datos?

 - A. mongorestore.
 - B. mongodump.
 - C. backup.
 - D. mongod.

2. ¿Qué comando de MongoDB permite recuperar una base de datos a partir de una copia de seguridad?

 - A. mongorestore.
 - B. save.
 - C. mongos.
 - D. copydb.

3. ¿Qué elementos de la base de datos mejoran el rendimiento de consultas a *collections*?

 - A. Índices.
 - B. *Replica sets*.
 - C. *Query routers*.
 - D. Las respuestas B y C son correctas.

4. ¿Qué característica de MongoDB permite tener redundancia y aumentar la disponibilidad de los datos?

 - A. Seguridad.
 - B. *Sharding*.
 - C. Índices.
 - D. Replicación.

5. ¿Cuál es el modelo básico de replicación en MongoDB?

- A. *Sharding*.
- B. *Replica set*.
- C. Maestro-Esclavo.
- D. *Shards*.

6. ¿Cómo se llama al refinamiento del modelo Maestro-Esclavo implementado en MongoDB?

- A. *Result set*.
- B. *Replica set*.
- C. *Sharding*.
- D. Replicación.

7. ¿Cómo se denomina el nodo de un *replica set* que no almacena datos y solamente puede votar en las elecciones de nodo primario?

- A. Secundario.
- B. Árbitro.
- C. *Shard*.
- D. *Config server*.

8. ¿Qué otro nombre recibe el método de escalabilidad horizontal, en el que los datos son separados y distribuidos entre varios servidores?

- A. Escalabilidad vertical.
- B. *Elastic computing*.
- C. *Sharding*.
- D. Replicación.

9. ¿Qué nombre reciben los nodos que almacenan datos en un *sharded cluster*?
 - A. *Data stores*.
 - B. *Config servers*.
 - C. *Query routers*.
 - D. *Shards*.
10. ¿Cuántos *config servers* debe haber en un entorno de producción?
 - A. Uno.
 - B. Tres.
 - C. Un máximo de cinco.
 - D. Depende del número de servidores disponibles.

Bases de Datos para el Big Data

Tema 6. Drivers de conexión

Índice

[Esquema](#)

[Ideas clave](#)

[6.1. Introducción y objetivos](#)

[6.2. Referencias: dónde consultar cada driver](#)

[6.3. Ejemplos de uso](#)

[A fondo](#)

[Python MongoDB Driver \(PyMongo\)](#)

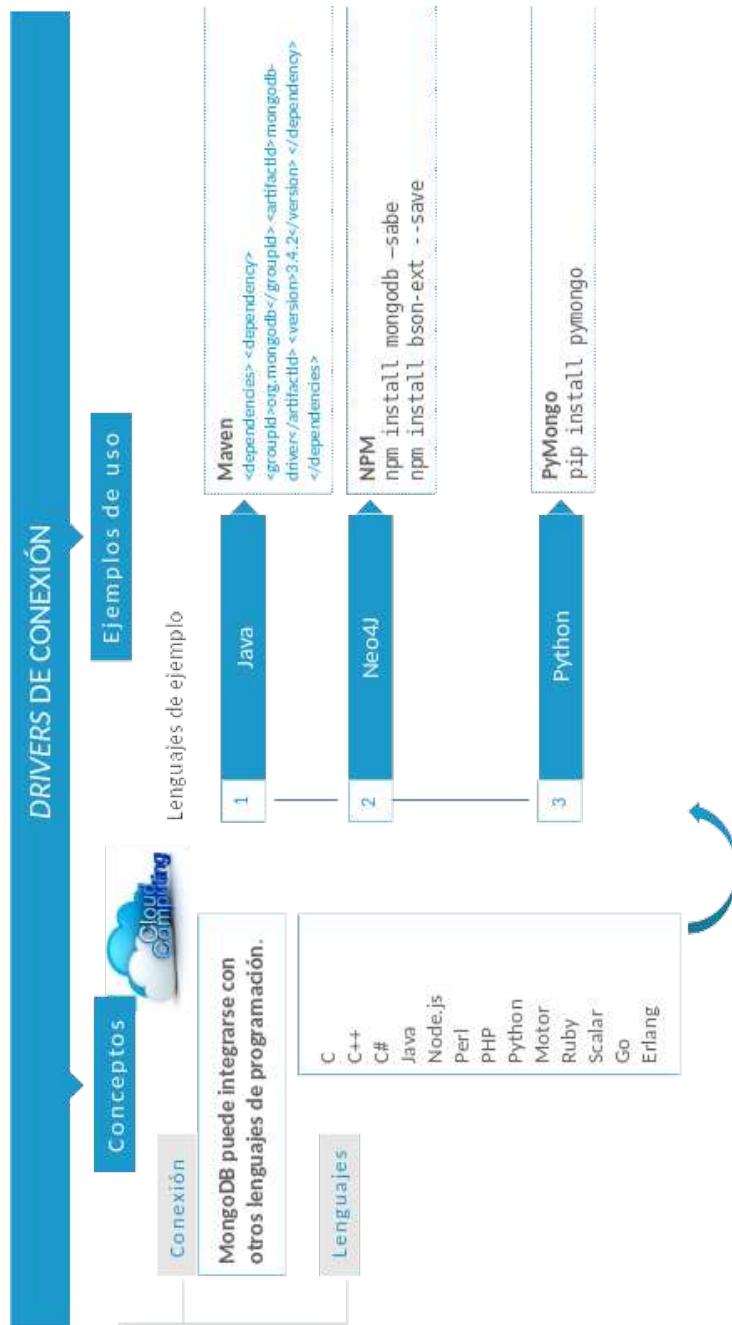
[Build a restful API with Node.js Express AND MongoDB.
Rest API tutorial](#)

[MongoDB Java Driver](#)

[MongoDB Node.js Driver](#)

[MongoDB Python Driver](#)

[Test](#)



6.1. Introducción y objetivos

Este tema pretende ser una pequeña guía para empezar a desarrollar aplicaciones utilizando MongoDB. Para estudiar este tema debes leer las Ideas clave y, si deseas información adicional sobre un concepto específico, puedes consultar los recursos sugeridos en A fondo. Se recomienda también intentar replicar los ejemplos y realizar las actividades para entender cada uno de los conceptos que se explican.

Para que un desarrollador pueda acceder a los datos almacenados en MongoDB desde el lenguaje de programación utilizado, es necesario utilizar un *driver* o controlador que nos haga de enlace con la base de datos. Es por esta razón que una vez conocido cómo consultar y gestionar los datos, es hora de conocer cómo poder utilizarlos desde cualquier programa que se quiera desarrollar, independientemente del lenguaje de desarrollo utilizado.

Existe un gran número de *drivers* que permiten trabajar con MongoDB, según el lenguaje de programación elegido. A lo largo de este tema se verán tres ejemplos de los tres más usados en aplicaciones de visualización y *big data*. Si se quiere empezar a desarrollar con algún otro, se proporciona un apartado de referencias que debe consultarse como punto de partida para conocer todos los *drivers* que MongoDB proporciona, así como complementar la información de los tres *drivers* tratados.

Para poder consultar, actualizar o insertar nuevos datos a una base de datos MongoDB a través de una aplicación programada en cualquier lenguaje de programación, es necesario tener un API que nos proporcione un enlace entre la base de datos y el código y nos permita utilizar aproximadamente las mismas funcionalidades que los utilizados desde la consola. Para ello, cada lenguaje de programación posee unos métodos o funciones que nos facilitan el desarrollo de nuestra aplicación, utilizando para ello una conexión transparente para el

programador. Estas API nos proporcionan suficientes funcionalidades para tener un acceso total a los datos.

Como cualquier otra API, cuando se está desarrollando un programa que debe acceder a una base de datos, lo que se busca es que las interfaces de conexión a dichas bases de datos sean transparentes para el programador. En definitiva, se encarga de «traducir» las llamadas que se hacen desde un lenguaje de programación a un «lenguaje» que entienda la base de datos.

Los objetivos que se persiguen con este tema son los siguientes:

- ▶ Conocer los diferentes *drivers* que existen para MongoDB.
- ▶ Aprender el uso de los *drivers* más populares para usar MongoDB desde otras aplicaciones.

6.2. Referencias: dónde consultar cada driver

MongoDB proporciona una descripción completa de cómo funciona cada uno de los *drivers*, que tienes disponible en el siguiente [enlace](#). En esta página se pueden encontrar cursos, tutoriales y ejemplos para facilitar la construcción de aplicaciones que precisen de MongoDB para su funcionamiento.

A continuación, se enumera una lista de los *drivers* que hay disponibles:

- ▶ C
- ▶ C++
- ▶ C#
- ▶ Java
- ▶ Node.js
- ▶ Perl
- ▶ PHP
- ▶ Python
- ▶ Motor
- ▶ Ruby
- ▶ Scalar
- ▶ Go
- ▶ Erlang

Además, la comunidad proporciona soporte para los siguientes *drivers*:

- ▶ Action Script3

- ▶ C

- ▶ C# and .NET

- ▶ Clojure

- ▶ ColdFusion

- ▶ D

- ▶ Dart

- ▶ Delphi

- ▶ Elixir

- ▶ Entity

- ▶ Erlang

- ▶ Factor

- ▶ Fantom

- ▶ F#

- ▶ Go

- ▶ Groovy

- ▶ Haskell

- ▶ JavaScript

- ▶ LabVIEW
- ▶ Lisp
- ▶ Lua
- ▶ Mathematica
- ▶ MatLab
- ▶ Node.js
- ▶ Objetive C
- ▶ OCaml
- ▶ Opa
- ▶ Perl
- ▶ PHP
- ▶ PowerShell
- ▶ Prolog
- ▶ Python
- ▶ R
- ▶ Ruby
- ▶ Scala
- ▶ Swift
- ▶ Racket

- ▶ Smalltalk

También MongoDB proporciona un conector para Hadoop. Este conector es un *plugin* que permite utilizar MongoDB como fuente de entrada y/o salida destino.

6.3. Ejemplos de uso

Java

Para poder utilizar MongoDB en Java, es necesario bajarse las librerías que permiten utilizar la versión requerida. Para ello, MongoDB proporciona varios métodos. Los comandos para bajar las dependencias con Maven son los siguientes:

```
<dependencies> <dependency> <groupId>org.mongodb</groupId>
<artifactId>mongodb-driver</artifactId> <version>3.4.2</version>
</dependency> </dependencies>
```

Otra alternativa es bajar los *.jar* de la fuente:

<https://oss.sonatype.org/content/repositories/releases/org/mongodb/mongo-java-driver/3.4.2/>

Para realizar los primeros ejemplos se recomienda descargar los siguientes archivos:

- ▶ mongo-java-driver-3.4.2-javadoc.jar : incluye la documentación.
- ▶ mongo-java-driver-3.4.2-source.jar : incluye las fuentes.
- ▶ mongo-java-driver-3.4.2.jar : incluye los diferentes métodos.

Por defecto el *driver* de Java se conecta a *localhost* por el puerto 27017. Para conectarse a la base de datos con los parámetros por defecto:

```
MongoClient mongoClient = new MongoClient();
```

Si la base de datos está en un *host* específico:

```
MongoClient mongoClient = new MongoClient( "host1" );
```

Si está en un host específico con un puerto específico:

```
MongoClient mongoClient = new MongoClient(new  
MongoClientURI("mongodb://host1:27017"));
```

Una vez obtenido el objeto que conecta con MongoDB es necesario acceder a una base de datos específica:

```
MongoDatabase database = mongoClient.getDatabase("test");
```

Ya solo falta acceder a la colección:

```
MongoCollection<Document> collection =  
database.getCollection("estudiantes");
```

Para ello, primero hay que conocer que en Java es necesario construir los documentos con una estructura específica. Si se tiene como ejemplo el siguiente documento:

```
{  
"marca" : "BMW",  
"puertas" : [3, 5]  
}
```

En formato BSON de Java se crearía de la siguiente forma:

```
new BsonDocument().append("marca", new BsonString("BMW"))  
    .append("puertas", new BsonArray(Arrays.asList(new  
BsonInt32(3), new BsonInt32(5))));
```

Por lo que una sencilla búsqueda se realizaría de la siguiente forma:

```
collection.find(new Document("marca", "BMW"))
```

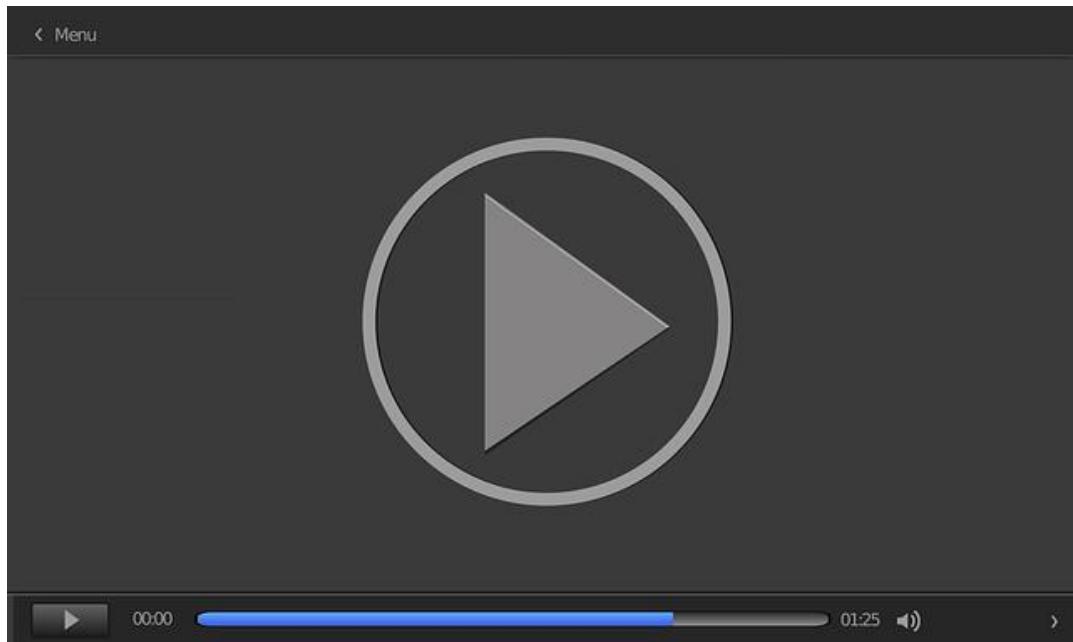
Si, por el contrario, lo que se pretende es insertar un documento, se podrá utilizar los métodos `insertOne()` e `insertMany()` que aceptan un documento o una lista de ellos. Un ejemplo de `insertOne` es el siguiente:

```
Document document = new Document("marca", "Audi")
    .append("motor", new Document("CC", 3000).append("CV", 210))
    .append("modelo", "A5")
    .append("colores", Arrays.asList("rojo", "negro", "blanco"));

collection.insertOne(document);
```

En la clase «Introducción del *driver Java para MongoDB*» se describe la instalación y breve introducción del *driver Java para MongoDB*. En este vídeo se enumeran los pasos para descargar e instalar el *driver Java de MongoDB* para realizar una aplicación Java con Eclipse.

También encontrarás un breve ejemplo de cómo se puede realizar una conexión a la base de datos, así como realizar consulta y trabajar con los resultados de la *query*.



Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=2c0c3063-27d2-430a-97dc-adbe00e851a2>

Vídeo 1. Introducción del *driver Java* para MongoDB.

Node.js

Para empezar, la forma más fácil de conseguir los *drivers* de Node.js 2.0 es usando **NPM** (*Node Package Manager*). Nada más crear el proyecto con **npm init**, se pueden instalar los *drivers* de MongoDB y sus dependencias con el siguiente comando:

```
npm install mongodb --save
```

Con ello se descargarán los *drivers* y se añadirá una dependencia en el archivo package.json. También se recomienda bajar la extensión **bson-ext**, ya que proporciona mejores formas de serialización y deserialización que el proporcionado por el parser JavaScript. Para instalar el *driver* es necesario ejecutar el siguiente comando:

```
npm install bson-ext --save
```

Para conectarse al servidor y la base de datos basta con añadir las siguientes líneas de código:

```
var MongoClient = require('mongodb').MongoClient
, assert = require('assert');
// Connection URL
var url = 'mongodb://localhost:27017/miProyecto';

// Use connect method to connect to the server
MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  console.log("Conexión realizada con éxito");
  db.close();
});
```

La aplicación imprimirá «Conexión realizada con éxito» si se han establecido una URL válida. Un ejemplo de inserción de datos es el que se puede ver a continuación:

```
var insertDocuments = function(db, callback) {  
    // Get the documents collection  
    var collection = db.collection('documents');  
    // Insert some documents  
    collection.insertMany([  
        {a : 1}, {a : 2}, {a : 3}  
    ], function(err, result) {  
        assert.equal(err, null);  
        assert.equal(3, result.result.n);  
        assert.equal(3, result.ops.length);  
        console.log("Inserted 3 documents into the collection");  
        callback(result);  
    });  
}
```

El comando `insert` devuelve un objeto con los siguientes campos:

- ▶ **result**: contiene el documento resultado de MongoDB.
- ▶ **ops**: contiene el documento insertado con el campo `_id` añadido.
- ▶ **connection**: contiene la conexión utilizada para llevar a cabo el `insert`.

Para llamar a la función `insertDocument` es necesario añadir el siguiente código:

```
var MongoClient = require('mongodb').MongoClient
, assert = require('assert');
// Connection URL
var url = 'mongodb://localhost:27017/myproject';
// Use connect method to connect to the server
MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  console.log("Connected successfully to server");
  insertDocuments(db, function() {
    db.close();
  });
});
```

Si se ejecuta la aplicación se obtendrá el siguiente resultado:

```
Connected successfully to server
Inserted 3 documents into the collection
```

Si se pretende realizar una búsqueda que muestre los documentos con el campo `a` igual a 3 se debería utilizar un código como el siguiente:

```
var findDocuments = function(db, callback) {
  // Get the documents collection
  var collection = db.collection('documents');
  // Find some documents
  collection.find({'a': 3}).toArray(function(err, docs) {
    assert.equal(err, null);
    console.log("Found the following records");
    console.log(docs);
    callback(docs);
  });
}
```

Un ejemplo para actualizar un conjunto de datos es el siguiente:

```
var updateDocument = function(db, callback) {
  // Get the documents collection
  var collection = db.collection('documents');
  // Update document where a is 2, set b equal to 1
  collection.updateOne({ a : 2 }
    , { $set: { b : 1 } }, function(err, result) {
    assert.equal(err, null);
    assert.equal(1, result.result.n);
    console.log("Updated the document with the field a equal to 2");
    callback(result);
  });
}
```

Y, para terminar, un ejemplo de borrado sería el siguiente:

```
var removeDocument = function(db, callback) {
  // Get the documents collection
  var collection = db.collection('documents');
  // Delete document where a is 3
  collection.deleteOne({ a : 3 }, function(err, result) {
    assert.equal(err, null);
    assert.equal(1, result.result.n);
    console.log("Removed the document with the field a equal to 3");
    callback(result);
  });
}
```

Python (PyMongo)

Para instalar **PyMongo** en Linux lo mejor es utilizar pip:

```
pip install pymongo
```

Y en Windows instalar PyMongo del siguiente enlace:

<https://pypi.python.org/pypi/pymongo/>

Para importar MongoClient desde PyMongo es necesario ejecutar el siguiente comando:

```
Import MongoClient from pymongo
```

Para crear una conexión utilizamos el cliente `MongoClient()`, que se conectará a *localhost* y al puerto 27017 si no se especifica ningún argumento. A continuación, se muestran unos ejemplos de conexión:

```
client = MongoClient()  
client = MongoClient("mongodb://mongo.example.es:27017")
```

Para acceder al objeto de base de datos se pueden utilizar cualquiera de las dos formas que se muestran a continuación:

```
db = client.myDB  
db = client['myDb']
```

Para acceder a una colección se procede de la siguiente forma:

```
coll = db.myCollection  
coll = db['myCollection']
```

Si se quiere insertar un documento:

```
from datetime import datetime
result = db.coches.insert_one(
{
    "coche": {
        "marca": "BMW",
        "Modelo": "430d",
        "CV": "220",
        "colores": ["rojo", "negro"]
    }
})
```

Para realizar una búsqueda:

```
cursor = db.coches.find({"marca": "BMW"})
for document in cursor:
    print(document)
```

Para actualizar un documento:

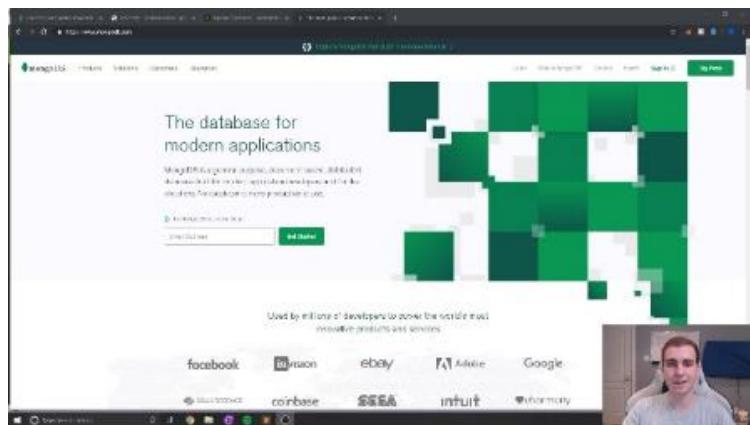
```
result = db.coches.update_one(
    {"marca": "BMW"},
    {
        "$set": {
            "CC": "3000"
        }
    }
)
```

Y, por lo tanto, para borrar un documento:

```
result = db.coches.delete_many({"marca": "BMW"})
```

Python MongoDB Driver (PyMongo)

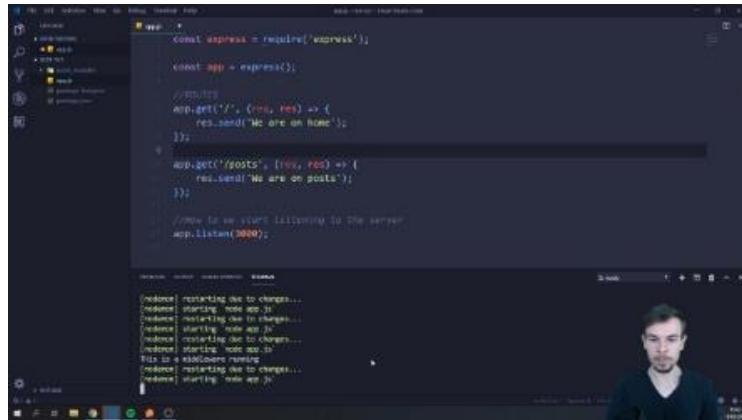
Tech With Tim. (21 de junio de 2019). *Python MongoDB Tutorial using PyMongo* [Archivo de vídeo]. https://www.youtube.com/watch?v=rE_bJl2GAY8



Este tutorial te mostrará cómo usar MongoDB con Python y el módulo de Python PyMongo. PyMongo es la API oficial de MongoDB que te permite realizar fácilmente operaciones de *database*.

Build a restful API with Node.js Express AND MongoDB. Rest API tutorial

Dev Ed. (11 de mayo de 2019). *Build A Restful Api With Node.js Express & MongoDB / Rest Api Tutorial* [Archivo de vídeo]. <https://www.youtube.com/watch?v=vjf774RKrLc>

A screenshot of a video player interface. On the left, there's a file explorer showing a folder structure with files like 'index.js' and 'package.json'. In the center, a code editor displays a Node.js script. The script starts with 'const express = require('express');' and defines two routes: one for '/' returning 'We are on home' and one for '/posts' returning 'We are on posts'. It ends with 'app.listen(3000);'. Below the code editor is a terminal window showing the output of running the script. The logs include messages like 'reloading due to changes...', 'restarting node app...', 'starting node app...', 'reloading due to changes...', 'restarting node app...', and 'This is a node.js app!'. On the right side of the video player, there's a small video thumbnail of a man with a beard.

Las APIS:REST ayudan a desacoplar nuestro código de *backend* de nuestro *front-end* para que podamos usarlo en múltiples aplicaciones (aplicaciones móviles, aplicaciones web, etc.). En este vídeo vas a aprender cómo construir una simple publicación de blog tipo api con todos los métodos útiles (GET, POST, DELETE, PATCH). Se usa Node.js como lenguaje de *backend*, express.js ayuda a crear rutas más fáciles. MongoDB junto con mongoose se usan para crear esquemas y modelos que definen cómo se ven nuestros datos.

MongoDB Java Driver

MongoDB. (2021). MongoDB Java Drivers [Página web]. <http://mongodb.github.io/mongo-java-driver/>

En este apartado se puede encontrar toda la información relativa a todos los *drivers* de Java, según la versión de MongoDB utilizada. En este enlace oficial de Mongo, además de explicar cómo funciona el *driver* y brindar multitud de ejemplos, también te enseña a utilizar diferentes librerías que facilitarán el desarrollo de aplicaciones en Java utilizando MongoDB.

MongoDB Node.js Driver

MongoDB. (2015). MongoDB Node.JS Driver [Página web] <https://mongodb.github.io/node-mongodb-native/>

En este apartado se puede encontrar toda la información relativa a todos los *drivers* de Node.js, según la versión de MongoDB utilizada. El enlace proporciona todos los ejemplos necesarios para empezar a desarrollar aplicaciones con MongoDB, además de dar recomendaciones sobre librerías que harán la vida del desarrollador más fácil.

MongoDB Python Driver

MongoDB. (2021). MongoDB Python Drivers [Página web]. <https://docs.mongodb.com/drivers/python>

En este apartado se pueden encontrar todos los enlaces necesarios para iniciarse en la programación de Python utilizando la base de datos MongoDB. Ofrece toda la información relativa a las versiones de MongoDB y una serie de tutoriales de desarrollo con varios ejemplos.

1. ¿Cuál es el objetivo de un *driver*?

 - A. «Traducir» las llamadas que se hacen desde un lenguaje de programación a un «lenguaje» que entienda la base de datos.
 - B. Proporcionar un objeto de conexión.
 - C. Proporcionar una serie de «funciones» que permitan al programador interactuar con la base de datos.
 - D. Todas las anteriores son correctas.
2. ¿Dónde se debería acudir si se quiere desarrollar una aplicación con base de datos MongoDB para gestionar una conexión?

 - A. A la página oficial del lenguaje de programación con el que estamos desarrollando.
 - B. A la página de documentación oficial de MongoDB, en el apartado de *drivers*.
 - C. A la página oficial del sistema operativo donde estemos desarrollando.
 - D. Todas las anteriores son correctas.
3. ¿Cuál de los siguientes lenguajes de programación soporta MongoDB?

 - A. C.
 - B. Java.
 - C. PHP.
 - D. Todos los anteriores son correctos.
4. ¿Cuál es la mejor forma de descargar el *driver* de Java?

 - A. Utilizando Maven.
 - B. Buscando en Google.
 - C. Repositorio.
 - D. Las respuestas A y C son correctas.

5. Si no se especifica en el *driver* ningún parámetro, ¿dónde se realiza la conexión?
 - A. *localhost* y puerto 27017.
 - B. Puerto 27017.
 - C. *Localhost*.
 - D. Es obligatorio definir un servidor y un puerto.
6. ¿Cuál es el formato de documentos utilizado por el *driver* de Java?
 - A. BSON.
 - B. JSON.
 - C. CSV.
 - D. XML.
7. ¿Cuál es la mejor forma de instalar el *driver* de Node.js?
 - A. Usando Maven.
 - B. Buscando en Google.
 - C. Usando NPM.
 - D. Todas las anteriores son correctas.
8. ¿Qué es PyMongo?
 - A. Una base de datos NoSQL.
 - B. El *driver* de MongoDB para Python.
 - C. Una base de datos SQL.
 - D. El *driver* de lenguaje de programación Py.

9. ¿Cómo podemos descargar el *driver* de Python para Mongo?
- A. Utilizando NPM.
 - B. Utilizando Maven.
 - C. Buscando en Google.
 - D. Utilizando PIP.
10. ¿Cuál de estos comandos son válidos para acceder a una colección de mongo?
- A. db.myCollection
 - B. db['myCollection']
 - C. db.getCollection('myCollection')
 - D. Todas son correctas.

Bases de Datos para el Big Data

Tema 7. Cassandra

Índice

Esquema

Ideas clave

- 7.1. Introducción y objetivos
- 7.2. Descarga e instalación
- 7.3. Conceptos generales
- 7.4. Modelo de datos y relaciones
- 7.5. CQL3
- 7.6. TTL y WriteTime
- 7.7. Índices
- 7.8. Ejercicio práctico: instalación con Docker
- 7.9. Referencias bibliográficas

A fondo

La documentación oficial de Apache Cassandra

Introducción a Cassandra con Datastax

Webinar sobre el modelado de datos en Cassandra

Instalar Cassandra en Windows

Bibliografía

Test

Apache Cassandra	
Definición	Generalidades
CQL3	Keyspaces, tablas y columnas
<p>Arquitectura:</p> <ul style="list-style-type: none"> - Cassandra está diseñada para trabajar en entornos <i>big data</i> sobre múltiples nodos evitando puntos de fallo. - Dispone de un sistema distribuido de igual a igual en todos sus nodos donde los datos se distribuyen entre todo el clúster. - Los nodos de un clúster pueden aceptar solicitudes de lectura y escritura, independientemente de dónde se encuentren realmente los datos en dicho clúster. - Si un nodo falla, las solicitudes de lectura/escritura pasan a ser atendidas por otros nodos de la red. 	<p>Manipulación de datos</p> <ul style="list-style-type: none"> • Tipos de datos. • Contadores. • Uso de <i>timestamp</i>. • Uso de <i>date</i>. • Definir duraciones. • Colecciones: <i>maps</i>, <i>sets</i> y <i>list</i>. • Tuplas. • UDT. • Operaciones de <i>INSERT</i>, <i>UPDATE</i>, <i>DELETE</i>.
<p>Modelo de datos:</p> <ul style="list-style-type: none"> - Los patrones de acceso a los datos y las consultas de la aplicación determinan la estructura y organización de los datos que luego se utilizan para diseñar las tablas de la base de datos. - Cada consulta está respaldada por una tabla y, por ello, los datos se duplican en varias tablas en un proceso conocido como desnormalización. - La duplicación de datos y un alto rendimiento de escritura se utilizan para lograr un alto rendimiento de lectura. 	<p>Otras funcionalidades</p> <ul style="list-style-type: none"> • TTL. • WriteTime. • Índices.
<p>Base de datos NoSQL de tipo clave-valor</p> <p>Cómo aplica el teorema CAP</p>	<p>Conocer los principales componentes de Cassandra y describir el sistema de replicación, la lógica de las operaciones de escritura y las de lectura.</p>
<p>Instalación sobre Linux y Windows</p>	

7.1. Introducción y objetivos

Este tema es totalmente práctico y se enfoca en el uso básico de la base de datos Cassandra, dentro del ámbito de NoSQL. Se describen los principales pasos de su instalación, una sección dedicada a conceptos generales, una sección que habla sobre el modelado de datos en esta base de datos y el resto de los temas se enfocará en aprender a configurar y manipular la base de datos.

Al tratarse de un tema con un enfoque práctico, la mejor forma de estudiarlo es llevando a cabo la instalación de la base de datos en un entorno local para pruebas. Para ello, se recomienda que se estudien inicialmente los apartados 7.2 y 7.8. Una vez instalada Cassandra en local, el estudiante podrá continuar con los apartados siguientes, según el orden propuesto y las indicaciones o sugerencias que brinde el profesor a lo largo de la asignatura.

Apache Cassandra es una base de datos NoSQL caracterizada por ser distribuida, altamente escalable y con un alto rendimiento. Su diseño y arquitectura distribuida le permiten manejar grandes volúmenes de datos sobre varios servidores, proporcionando así una alta disponibilidad con el mínimo número de fallos. Para lograr la alta disponibilidad, los datos se colocan en diferentes máquinas con más de un factor de replicación determinado.

Cassandra es un producto de Apache y su sistema de almacenamiento o motor de base de datos es de código abierto, distribuido y descentralizado. Su fin principal es gestionar grandes volúmenes de datos estructurados, repartidos por distintos servidores.

¿Qué se puede destacar de Cassandra?

- ▶ Es una base de datos orientada a columnas.

- ▶ Es escalable, consistente y tolerante a fallas.
- ▶ El diseño de distribución de Cassandra se basa en Dynamo de Amazon y su modelo de datos en Bigtable de Google.
- ▶ Se puede decir que Facebook es el creador de Cassandra.
- ▶ Como motor de base de datos, es totalmente diferente de los sistemas de administración de bases de datos relacionales.
- ▶ El modelo de replicación es del estilo de Dynamo y el concepto de modelo de datos «familia de columnas» es el más robusto.
- ▶ ¿Se usa? Sí. Algunas de las empresas más grandes como Facebook, Twitter, Cisco, Rackspace, eBay, Twitter y Netflix, entre otros, lo emplean dentro de sus arquitecturas.

Los objetivos que se abordan en este tema son los siguientes:

- ▶ Entender los conceptos generales que caracterizan a Cassandra.
- ▶ Conocer la arquitectura básica de esta base de datos NoSQL y los elementos que la componen.
- ▶ Aprender a usar los principales tipos de datos, incluidos los tipos de datos definidos por el usuario.
- ▶ Entender el uso de las colecciones *set*, *list* y *map*.
- ▶ Poner en práctica la instalación de esta base de datos en un entorno de pruebas.

7.2. Descarga e instalación

La instalación propuesta se hará sobre distribuciones de Linux Debian o Ubuntu.

Para descargar los ficheros de instalación correspondientes, ten en cuenta que:

- ▶ Para *<release series>* se debe especificar el número de versión principal sin punto y con una x.
- ▶ La última *<release series>*, a fecha de noviembre de 2020, es la 311x.
- ▶ Para versiones anteriores, el *<release series>* puede ser uno de 30x, 22x o 21x.
- ▶ Agrega el repositorio Apache de Cassandra /etc/apt/sources.list.d/cassandra.sources.list , por ejemplo, para la versión 3.11:

```
$ echo "deb https://downloads.apache.org/cassandra/debian 311x main" | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list
```

- ▶ Agrega las claves del repositorio de Apache Cassandra y, acto seguido, actualiza los repositorios:

```
$ curl https://downloads.apache.org/cassandra/KEYS | sudo apt-key add -
$ sudo apt-get update
```

- ▶ Agrega la clave pública correspondiente y vuelva a actualizar los repositorios de la siguiente manera:

```
$ sudo apt-key adv --keyserver pool.sks-keyservers.net --recv-key
A278B781FE4B2BDA
$ sudo apt-get update
```

La clave pública puede ser diferente, es posible obtenerla o bien del mensaje de error o bien a través de la lista completa de claves públicas de los contribuyentes de Apache, visita la siguiente URL: <https://downloads.apache.org/cassandra/KEYS>.

- ▶ Instala ahora Cassandra desde la consola:

```
$ sudo apt-get install cassandra
```

Ten en cuenta que para iniciar el servicio o demonio de Cassandra, puedes utilizar la instrucción `sudo service cassandra start` . Para detenerlo, `sudo service cassandra stop` .

Después de la instalación, el demonio se iniciará automáticamente. Si necesitas hacer algún cambio en la configuración de la base de datos, asegúrate de detenerlo antes para que los cambios se apliquen correctamente. Comprueba que Cassandra se está ejecutando, utilizando el comando `nodetool status` desde la línea de comando.

Para iniciar la consola `cqlsh`, simplemente escribe este mismo comando: `$ cqlsh`

Diretorios de instalación:

- ▶ **/etc/cassandra**: ubicación predeterminada de los archivos de configuración.
- ▶ **/var/log/cassandra/** y **/var/lib/cassandra**: ubicación predeterminada de los directorios de datos y registros respectivamente.
- ▶ **/etc/default/cassandra**: aquí se encuentran las opciones de inicio que se pueden configurar.

El código fuente de Cassandra está disponible en Git, consulta la URL:

<https://gitbox.apache.org/repos/asf/cassandra.git>.

7.3. Conceptos generales

En el año 2008 Cassandra fue desarrollada en Facebook, surgió como una mezcla del almacén de datos BigTable utilizado por Google y el almacén de datos Dynamo utilizado por Amazon (Carpenter y Hewitt, 2020). Su creador fue Avinash Lakshman, autor también de Amazon Dynamo, y Prashant Malik (Carpenter y Hewitt, 2020; Neeraj, 2013).

Con esta base de datos se quería resolver el problema de búsqueda en la bandeja de entrada de Facebook. Actualmente ha evolucionado mucho y su uso se extiende notablemente.

Las características principales de Cassandra son:

- ▶ Cassandra es una base de datos de tipo clave-valor.
- ▶ Los datos se almacenan como tablas y columnas.
- ▶ Cada tabla tiene una clave principal.
- ▶ Aunque permite lecturas y escrituras muy rápidas, tiene una interfaz SQL limitada debido a su propia estructura.

Arquitectura

Como se mencionó anteriormente, Cassandra se diseñó para trabajar en entornos *big data* sobre múltiples nodos, evitando puntos de fallo por caídas de servidores o nodos. Esta característica la consigue porque cuenta con un sistema distribuido de igual a igual en todos sus nodos, donde los datos se distribuyen entre todo el clúster (Neeraj, 2013). Aunque cada nodo es independiente, al mismo tiempo está interconectado con otros nodos, desempeñando el mismo papel que los demás (Kan, 2014).

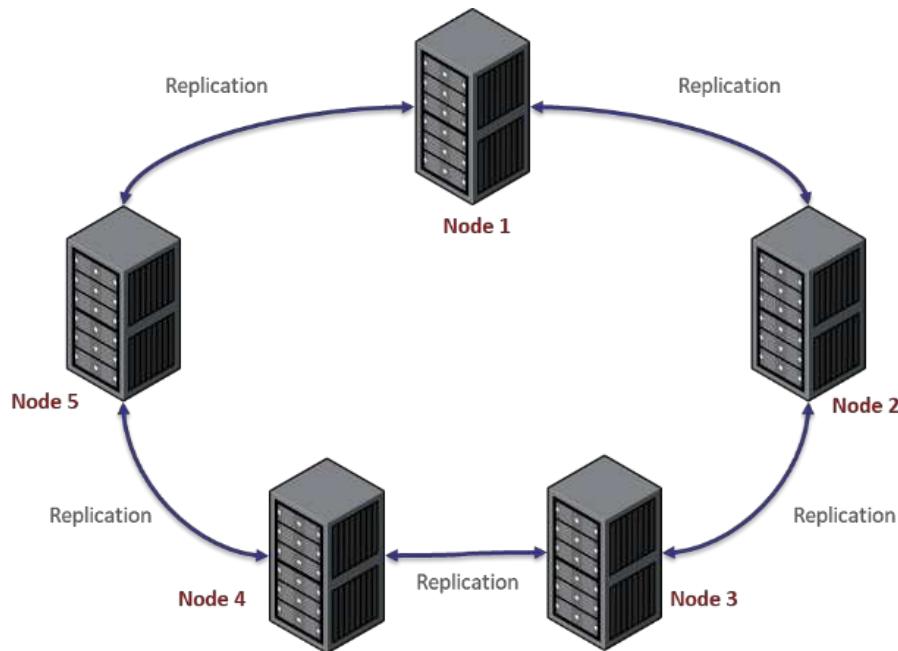


Figura 1. Modelo de referencia arquitectura Apache Cassandra.

Los nodos de un clúster pueden aceptar solicitudes de lectura y escritura, independientemente de dónde se encuentren realmente los datos en dicho clúster. Si un nodo falla, las solicitudes de lectura y escritura pasan a ser atendidas por otros nodos de la red (Kan, 2014).

Los principales componentes de Cassandra son:

- ▶ **Nodo:** un lugar donde se almacenan los datos.
- ▶ **Centro de datos:** es una colección de nodos relacionados.
- ▶ **Clúster:** es un componente que contiene uno o más centros de datos.
- ▶ **Registro de confirmación:** mecanismo de recuperación de fallos. Todas las operaciones de escritura se escriben en este registro de confirmación.

- ▶ **Mem-table:** es una estructura de datos residente en memoria. Después del registro de confirmación, los datos pasan a la tabla en memoria. Es normal que para una familia de una sola columna existan varias mem-tables.
- ▶ **SSTable:** archivo de disco al que se vacían los datos de la tabla de memoria, cuando su contenido alcanza un valor umbral.
- ▶ **Filtro Bloom:** conjunto de algoritmos no deterministas muy rápidos, utilizados para probar si un elemento es miembro de un conjunto. Comprende el uso de tipo especial de caché al que se accede después de cada consulta.

Sistema de replicación

Los nodos de un clúster en Cassandra actúan como réplicas de un dato específico. Si la respuesta a algún nodo incluye un valor desactualizado, Cassandra se encargará de devolver al usuario el valor más reciente. Después de atender al usuario, Cassandra repara la lectura en segundo plano para actualizar los valores que se encontraron obsoletos.

Operaciones de escritura

Los registros de confirmación que residen en los nodos capturan toda actividad de escritura de los mismos nodos. El objetivo es que dichos datos sean almacenados en la tabla de memorias siempre que esta esté libre. En el caso de que la tabla de memorias esté llena, los datos se escribirán en el archivo de datos SSTable.

La partición y replicación de todas las escrituras se lleva a cabo de forma automática en todo el clúster. La consolidación de las SSTables se realiza periódicamente, descartando los datos considerados como innecesarios.

Operaciones de lectura

Para leer los datos, Cassandra consulta los valores de la tabla de memorias y establece la forma de ubicar la tabla SST que contenga los datos solicitados. Las operaciones de lectura se tipifican en solicitudes directas, de resumen y de

reparación antes de ser delegadas a los coordinadores, para que las envíen a las réplicas correspondientes.

El primer paso es enviar una solicitud directa a una de las réplicas. Luego, el coordinador envía la solicitud de resumen a la cantidad de réplicas especificadas para garantizar el nivel de coherencia y, finalmente, comprueba que los datos devueltos están actualizados.

Por último, el coordinador envía la solicitud de resumen a todas las réplicas restantes.

En el caso de que algún nodo devuelva un valor desactualizado, la solicitud de reparación de lectura en segundo plano actualizará dichos valores.

Lenguaje de consulta en Cassandra

Es necesario conocer también Cassandra Query Language (CQL), el lenguaje de consulta utilizado para acceder a Cassandra a través de sus nodos. La principal característica del lenguaje es que CQL utiliza la base de datos como un contenedor de tablas, las cuales representan espacio de claves. Los programadores pueden utilizar *cqlsh* en modo consola para trabajar con CQL o emplear controladores de otras aplicaciones independientes.

7.4. Modelo de datos y relaciones

El modelado de datos es el proceso de identificar entidades y sus relaciones. En los sistemas relacionales, los datos se almacenan en tablas normalizadas con claves externas que se utilizan para hacer referencia a los relacionados entre tablas (Sharma, 2014). Las consultas que realizará la aplicación están controladas por la estructura de las tablas y los datos relacionados se consultan como combinaciones de dichas tablas.

En Cassandra, el modelado de datos se basa en consultas. Los patrones de acceso a los datos y las consultas de la aplicación determinan la estructura y organización de los datos que luego se utilizan para diseñar las tablas de la base de datos. A diferencia de un modelo de base de datos relacional en el que las consultas utilizan combinaciones de tablas para obtener datos de varias tablas, las combinaciones no son compatibles con Cassandra, por lo que todos los campos obligatorios (columnas) deben agruparse en una sola tabla.

Dado que cada consulta está respaldada por una tabla, los datos se duplican en varias tablas en un proceso conocido como desnormalización. La duplicación de datos y un alto rendimiento de escritura se utilizan para lograr un alto rendimiento de lectura.

La elección de la clave principal y la clave de partición es importante para distribuir los datos de manera uniforme en todo el clúster. Mantener el número de particiones leídas para una consulta al mínimo también es importante, porque diferentes particiones podrían estar ubicadas en diferentes nodos y el coordinador necesitaría enviar una solicitud a cada nodo agregando a la solicitud la sobrecarga y la latencia.

Cassandra divide los datos en los nodos de almacenamiento, utilizando una variante de *hash* consistente para la distribución de datos. El *hash* es una técnica que se

utiliza para mapear datos con la que, dada una clave, una función *hash* genera un valor *hash* (o simplemente un *hash*) que se almacena en una tabla *hash*. Una clave de partición se genera a partir del primer campo de una clave primaria.

Los datos divididos en tablas *hash* mediante claves de partición proporcionan una búsqueda rápida. Cuantas menos particiones utilizadas para una consulta, más rápido es el tiempo de respuesta de la consulta. Sabiendo lo anterior, ahora es posible conocer los componentes que describen un modelo de datos en Cassandra. Dichos componentes son espacios de claves, tablas y columnas.

Espacios de claves

El modelo de datos de Cassandra consta de espacios de claves al más alto nivel. Los espacios de claves son los contenedores de datos, similares al esquema o la base de datos en una base de datos relacional. Normalmente, los espacios de claves contienen muchas tablas.

Tablas

Dentro de los espacios de claves, se definen o crean las tablas. Estas también se conocen como «familias de columnas» en las versiones anteriores de Cassandra. Las tablas contienen un conjunto de columnas y una clave principal, y almacenan los datos en un conjunto de filas.

Columnas

Las columnas definen la estructura de los datos en una tabla. Cada columna tiene un tipo asociado, como entero, texto, doble y booleano.

A continuación, se muestran ejemplos de tablas en Cassandra.

EJEMPLO DE CREACIÓN DE TABLAS	
<pre>CREATE TABLE tab1 (id int, age int, name text, PRIMARY KEY (id));</pre>	<pre>CREATE TABLE tab2 (id int, name text, age int, dir text, PRIMARY KEY (id,name));</pre>

Tabla 1. Código de ejemplo para la creación de tablas en Cassandra.

En la primera columna de la tabla anterior, la clave de partición se genera a partir de la clave principal `id` para la distribución de datos entre los nodos de un clúster. En la segunda columna de la tabla, se asigna una clave primaria compuesta, el primer campo `id` se usa para generar la clave de partición y el segundo campo `name` es la clave de agrupamiento usada para ordenar dentro de una partición.

El uso de claves de agrupación en clústeres para ordenar datos hace que la recuperación de datos adyacentes sea más eficiente.

Relaciones

Cuando se quiere crear una aplicación basada en datos que utilizarán una base de datos relacional, es posible comenzar a modelar el dominio como un conjunto de tablas normalizadas correctamente y utilizar claves externas para hacer referencia a datos relacionados en otras tablas.

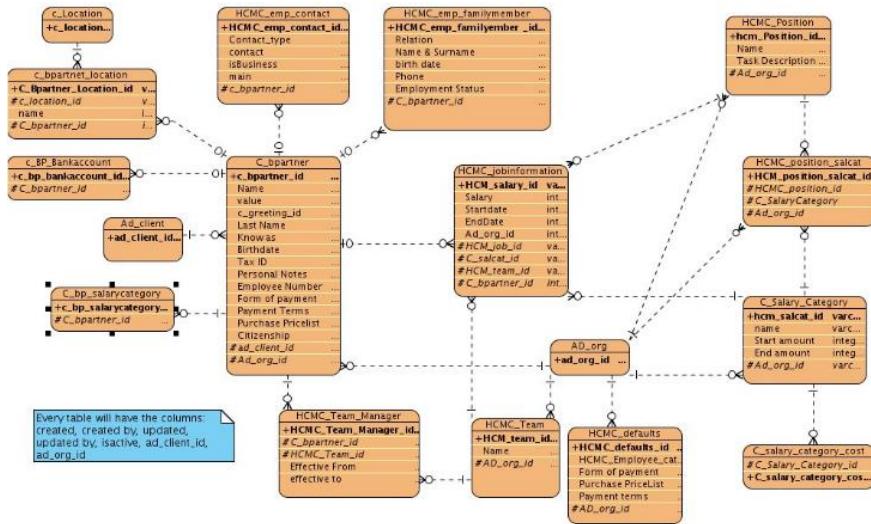


Figura 2. Ejemplo de una base de datos relacional.

Al utilizar Cassandra no es posible realizar **uniones**. Si tu necesidad es combinar datos, tendrás que crear una segunda tabla desnormalizada que represente los resultados de la combinación para ti. Esta es la filosofía de Cassandra, la cual, además, no aplica integridad referencial y destaca por el uso de la desnormalización de los datos. Estos son los aspectos que nos llevan a pensar en el diseño de consulta típico en Cassandra.

El diseño de la interfaz de la aplicación suele ser muy útil para comenzar a identificar consultas. Supón que has hablado con las partes interesadas del proyecto y que tus diseñadores de UX han producido diseños de interfaz de usuario o *wireframes* para los casos de uso clave.

Algunas consultas de ejemplo podrían ser:

- ▶ Q1. Busca parques cerca del hotel del cliente.
- ▶ Q2. Busca información sobre una cafetería concreta de la zona.
- ▶ Q3. Encuentra puntos de interés cercanos al hotel del cliente.

- ▶ Q4. Encuentra un punto de información turístico de la zona.
- ▶ Q5. Encuentra la disponibilidad del museo en unas fechas y horas concretas.



Figura 3. Ejemplo de esquema para el diseño de consultas en Apache Cassandra.

7.5. CQL3

Como se ha visto en los apartados anteriores, los términos utilizados para referirse a la base de datos Cassandra son los mismos de SQL (tablas, filas y columnas). Para manipular los datos alojados en estos elementos, se utiliza CQL como un lenguaje sutilmente parecido a SQL. CQL3 corresponde a la versión 3 del lenguaje, el cual no es compatible con las versiones CQL1 y CQL2.

Tipos de datos

CQL3 soporta los tipos de datos nativos, de colección, definidos por el usuario y personalizados. Los siguientes son algunos ejemplos de los tipos de datos nativos.

::= ASCII	ascii	string
BIGINT	bigint	integer
BLOB	blob	blob
BOOLEAN	boolean	boolean
COUNTER	counter	integer
DATE	date	integer , string
DECIMAL	decimal	integer , float
DOUBLE	double	integer float
DURATION	duration	duration ,
FLOAT	float	integer , float
INET	inet	string
INT	int	integer
SMALLINT	smallint	integer
TEXT	text	string
TIME		
TIMESTAMP		
TIMEUUID		
TINYINT		
UUID		
VARCHAR		
VARINT		

Figura 4. Ejemplos de tipos de datos en Cassandra.

Contadores

Son tipos de columnas que funcionan como contadores, cuyas únicas operaciones permitidas son incrementar y decrementar. El valor de la columna no se establece,

Cassandra se encarga de controlar su valor.

Timestamp

Son valores codificados como enteros de 64 bits con signo. Se ingresan usando el tipo *integer* o *string* para representar, por ejemplo, una fecha ISO 8601. Estos son ejemplos de valores válidos como *timestamp*:

- ▶ 1299038700000
- ▶ '2011-02-03 04:05+0000'
- ▶ '2011-02-03 04:05:00+0000'
- ▶ '2011-02-03 04:05:00.000+0000'
- ▶ '2011-02-03T04:05+0000'
- ▶ '2011-02-03T04:05:00+0000'
- ▶ '2011-02-03T04:05:00.000+0000'

Date

Al igual que los *timestamp*, se puede representar como *integer* o *string*. En el caso de *string* se debe utilizar el formato convencional *yyyy-mm-dd*. Para el caso de las horas, se utilizan los mismos tipos primitivos y los formatos comunes son:

- ▶ '08:12:54'
- ▶ '08:12:54.123'
- ▶ '08:12:54.123456'
- ▶ '08:12:54.123456789'

Duraciones

Los valores del *duration* se codifican como números enteros con 3 signos de longitudes variables. El primer número entero representa el número de meses, el segundo el número, de días y el tercero, el número de nanosegundos. La forma de establecer su valor es (*quantity unit*)+, de la forma *12h30m*.

- ▶ y: años (12 meses)
- ▶ mo: meses (1 mes)
- ▶ w: semanas (7 días)
- ▶ d: días (1 día)
- ▶ h: horas (3.600.000.000.000 nanosegundos)
- ▶ m: minutos (60.000.000.000 nanosegundos)
- ▶ s: segundos (1.000.000.000 nanosegundos)
- ▶ ms: milisegundos (1.000.000 nanosegundos)
- ▶ uso μ s: microsegundos (1000 nanosegundos)
- ▶ ns: nanosegundos (1 nanosegundo)

Otros métodos para establecer este valor es mediante el formato ISO 8601: P[n]Y[n]M[n]DT[n]H[n]M[n]S o P[n]W y el formato alternativo ISO 8601: P[YYYY]-[MM]-[DD]T[hh]:[mm]:[ss].

Algunos ejemplos de su uso se muestran a continuación:

```
INSERT INTO RiderResults (name, race, result) VALUES ('Andres', 'Tour de France', 89h4m48s);
INSERT INTO RiderResults (name, race, result) VALUES ('Maria', 'Tour de France', PT89H8M53S);
```

```
INSERT INTO RiderResults (name, race, result) VALUES ('Marco', 'Tour de France', P0000-00-00T89:09:09);
```

Colecciones

Los tres tipos de colecciones que admite CQL son **map**, **set** y **list**.

- ▶ **Map:** es un conjunto (ordenado) de pares clave-valor, donde las claves son únicas y el mapa está ordenado por sus claves.

```
CREATE TABLE clients (
    id text PRIMARY KEY,    name text,    type map<text, text> //Aqui define un Map con Clave-Valor Text-Text
);
```

```
INSERT INTO clients (id, name, type) VALUES ('client001', 'Andres Luna', {
'bill' : 'ordinary', 'payment' : 'cash' });

// Actualiza un valor dentro de Map
UPDATE clients SET type = {'payment': 'PayPal'} WHERE id = 'client001';
UPDATE clients SET type['payment'] = 'VISA' WHERE id = 'client001';
UPDATE clients SET type = type + {'mortage': 'Fixed Interest',
'investment': 'Risk'} WHERE id = 'client001';
DELETE type['mortage'] FROM clients WHERE id = 'client001';
UPDATE clients SET type = type - { 'mortage', 'investment'} WHERE id =
'client001';
```

- ▶ **Set:** es una colección ordenada de valores únicos.

```
CREATE TABLE clients (
    id text PRIMARY KEY,    name text,    type set<text> //Aqui define un Set de tipo Text
);
```

```
INSERT INTO clients (id, name, type) VALUES ('client002', 'Manuel Faena',
{'A', 'B', 'C'});

UPDATE clients SET type = {'E', 'P', 'M'} WHERE id = 'client002';
//actualizas todo el Set
UPDATE clients SET type = type + {'H', 'F'} WHERE id = 'client002'; // insertar un elemento que existe no es posible
```

```
UPDATE clients SET type = type - {'H'} WHERE id = 'client002'; //no es un error intentar borrar si no existe
```

- ▶ **List:** es una colección ordenada de valores no únicos donde los elementos se ordenan por su posición en la lista.

```
CREATE TABLE products (
    id text PRIMARY KEY, name text, order, pos list<int> //Aquí define una Lista de enteros
);
```

```
INSERT INTO products (id, name, order, pos) VALUES ('Table01', 'Table Reduce', 3, [10, 11, 12]);
```

```
UPDATE products SET position = [ 210, 111, 412] WHERE id = 'Table01';
//actualiza toda la lista
UPDATE products SET order = 5, pos = pos + [ 45, 78 ] WHERE id = 'Table01';
// insertar dos elemento en la lista
UPDATE products SET order = 4, pos = pos - [78] WHERE id = 'client002';
//no es un error intentar borrar si no existe
UPDATE products SET pos[1] = 25 WHERE id = 'client002';
DELETE pos[2] products WHERE id = 'client002';
```

- ▶ **Tuple:** es un conjunto de valores de longitud determinada donde cada valor puede ser de un tipo diferente.

```
CREATE TABLE users (
    user text,
    phones tuple<text, text>,
)
```

```
INSERT INTO users (user, phones) VALUES ('Pepe Navarro', ('home', '(34) 666-666-666'));
```

Tipos definidos por el usuario (UDT)

Los usuarios pueden crear sus propios tipos de datos. Para ello se emplea la instrucción CREATE TYPE.

```
CREATE TYPE phone (
    code int,
    number text,
```

```
)  
  
CREATE TYPE address (  
    street text,  
    city text,  
    zip text,  
    phones map<text, phone>  
)  
  
CREATE TABLE user (  
    name text PRIMARY KEY,  
    addresses map<text, frozen<address>>  
)
```

Ten en cuenta que, si intentas crear una UDT que existe,
obtendrás un error.

Por otra parte, a partir de la versión 3.7 de Cassandra, los UDT en la
mayoría de los casos deben congelarse (*frozen*).

Frozen cambia el comportamiento de una colección o UDT. Al congelar
una colección, los valores se serializan como un todo, impidiendo que
dicha colección se pueda actualizar parcialmente. Al aplicar *frozen*, la
colección se debe tratar como un todo en el momento de manipularla.

Keyspace

Es un objeto que contiene las familias de columnas, los índices, los UDT, el
conocimiento del centro de datos, la estrategia utilizada en el espacio de claves, el
factor de replicación, etc. En Cassandra, se puede considerar el *keyspace* como algo
similar a la base de datos en un RDBMS.

```
CREATE keyspace KeyspaceName with replication={'class' : strategy name,  
'replication_factor' : No of replications on different nodes};
```

```
//Ejemplo
CREATE keyspace Products with replication=
{'class':SimpleStrategy,'replication_factor': 3};

//Modificar un keyspace
ALTER keyspace KeyspaceName with replication={'class':'StrategyName',
    'replication_factor': no of replications on different nodes}
    with DURABLE_WRITES=true/false
//Ejemplo
ALTER Keyspace Products with replication=
{'class':'NetworktopologyStrategy', 'DataCenter1':1};
#Borrado
DROP keyspace Products
```

- ▶ **Estrategia simple (SimpleStrategy):** se utiliza cuando solo tiene un centro de datos. En esta estrategia, la primera réplica se coloca en el nodo seleccionado por el particionador. Los nodos restantes se colocan en el sentido de las agujas del reloj en el anillo sin tener en cuenta la ubicación del *rack* o del nodo.

- ▶ **Estrategia de topología de red (NetworkTopologyStrategy)**: se utiliza cuando tiene más de un centro de datos. En esta estrategia, se debe proporcionar un factor de replicación para cada centro de datos por separado. La estrategia de topología de red coloca réplicas en nodos en el sentido de las agujas del reloj en el mismo centro de datos. Esta estrategia intenta colocar réplicas en diferentes *racks*.
- ▶ **Factor de replicación (ReplicationFactor)**: es el número de réplicas de datos colocadas en diferentes nodos. Para que no haya fallas, 3 es un buen factor de replicación. Más de dos factores de replicación garantizan que no haya un solo punto de falla. A veces, el servidor puede estar inactivo o puede ocurrir un problema de red, luego otras réplicas brindan servicio sin fallas.

Ten en cuenta los siguientes aspectos sobre el *keyspace*:

- ▶ Nombre del *keyspace*: no se puede modificar en Cassandra.
- ▶ Nombre de la estrategia: se puede modificar especificando un nuevo nombre de estrategia.
- ▶ Factor de replicación: se puede modificar especificando un nuevo factor de replicación.
- ▶ DURABLE_WRITES : el valor DURABLE_WRITES se puede modificar especificando su valor verdadero o falso. Por defecto, es true . Si se establece en false , no se escribirán actualizaciones en el registro de confirmación y viceversa.

INSERT: insertando datos

El comando INSERT escribe datos en columnas Cassandra en forma de fila. Dicho comando almacenará solo aquellas columnas proporcionadas por el usuario. Debe especificar necesariamente solo la columna de clave principal. Esta instrucción no devuelve ningún resultado.

```
INSERT INTO Products.users (user, phones) VALUES ('Pepe Navarro', ('home', '(34) 666-666-666'));
```

Ten en cuenta que Cassandra insertará una fila si no existe la clave principal indicada; de lo contrario, si la clave principal existe, actualizará esa fila.

UPDATE: actualizando datos

El comando `UPDATE` se emplea para actualizar los datos en la tabla Cassandra. Si no se devuelven resultados después de actualizar los datos, significa que los datos se actualizaron correctamente; de lo contrario, se devolverá un error. Los valores de columna se cambian en la cláusula `Set`, mientras que los datos se filtran con la cláusula `WHERE`.

```
UPDATE Products.users Set phone = '(34) 666-666-666' WHERE user = 'Pepe Lopez';
```

DELETE: borrando datos

El comando `DELETE` elimina una fila completa o algunas columnas de la tabla indicada. Cuando se eliminan datos, no se eliminan de la tabla inmediatamente. En su lugar, los datos eliminados se marcan con una lápida y se eliminan después de la compactación.

```
DELETE from Products.users WHERE user = 'Pepe Lopez';
```

Limitaciones en CQL

- ▶ No admite consultas de agregación como `max`, `min`, `avg`.

- ▶ No admite **ni group by ni having**.
- ▶ No admite **joins**.
- ▶ No admite consultas OR.
- ▶ No admite consultas con comodines.
- ▶ No admite consultas de unión e intersección.
- ▶ Las columnas de la tabla no se pueden filtrar sin crear el índice.
- ▶ La consulta mayor que (>) y menor que (<) solo se admite en la columna de agrupación.
- ▶ CQL no es adecuado para realizar análisis porque presenta muchas limitaciones.

7.6. TTL y WriteTime

Cassandra puede hacer que los datos expiren de forma automática. Durante la inserción de datos, por ejemplo, se debe especificar el valor TTL en segundos. Dicho valor TTL es el tiempo de vida de los datos. Después que pase ese período en concreto, los datos se eliminarán automáticamente.

Supón que quieres insertar un registro y que **este exista** solo durante **los primeros** 200 segundos. Al insertar, si indicas el TTL 200, los registros se eliminarán automáticamente después de los 200 segundos. Recuerda que los datos no se borran de inmediato, cuando estos caduquen se marcarán con una lápida. Existe una lápida durante un período de gracia, una vez que los datos caducan, estos se eliminan después del proceso de compactación.

```
INSERT INTO keyspaceName.TableName (ColumnNames) values (ColumnValues)
using TTL TimeInSeconds;
#Ejemplo
INSERT INTO Products.users (user, phones) VALUES ('Pepe Navarro', ('home',
'(34) 666-666')) using TTL 100;
```

La función WRITETIME, por su parte, es muy útil en CQL para recuperar la fecha y hora de los insert en las columnas. Es posible usar la función WRITETIME junto con la instrucción select seguida de la columna sin particiones entre paréntesis. En CQL una tabla contiene el *timestamp* que representa la fecha y hora en que se produjo una escritura en una columna. El valor devuelto por la función WRITETIME está en microsegundos, luego es posible convertirlo en formatos de fecha/hora.

```
CREATE TABLE ftest (
Id int,
Name text,
Address text,
PRIMARY KEY(Id)
);
INSERT INTO ftest (Id, Name, Address) VALUES (201, 'Ashish', 'Delhi');
INSERT INTO ftest (Id, Name, Address) VALUES (202, 'Rana', 'Mumbai');
```

```
INSERT INTO ftest  (Id, Name, Address) VALUES (203, 'Abi', 'Noida');  
SELECT * FROM ftest;  
SELECT WRITETIME (Address) FROM ftest;
```

7.7. Índices

CQL permite la creación de índices secundarios en tablas, los cuales son usados por las consultas que utilicen dicha tabla. Un índice secundario se identifica por un nombre definido por el usuario al momento de crearlo. Los índices se crean con el comando `CREATE INDEX`, el cual se especifica por el usuario sobre la columna deseada.

Si los datos ya existen para la columna que se desea indexar, Cassandra crea índices sobre los datos, durante la ejecución de la declaración del comando anterior.

Después de crear un índice, Cassandra indexa datos nuevos automáticamente cuando se insertan.

El índice no se puede crear sobre una clave principal, porque está ya está indexada. Un aspecto a tener en cuenta es que Cassandra no admite índices sobre colecciones.

```
CREATE INDEX PhoneIndex on Products.users (Phone);
```

Si no se indexa la columna, Cassandra no podrá filtrar dicha columna a menos que sea una clave principal. Por lo anterior, para filtrar columnas en Cassandra, es necesario crear índices.

El comando `DROP INDEX` permite eliminar el índice que se especifique. Si al crear el índice no se le dio un nombre, entonces el nombre del índice es `TableName_ColumnName_idx`. Ten en cuenta que, si el índice no existe, devolverá un error a menos que si la cláusula `IF EXISTS` , la cual devolverá `no-op`.

Se recomienda que, durante la creación del índice, se debe especificar el nombre del espacio de claves con el nombre del índice; de lo contrario, el índice se eliminará del espacio de claves actual.

```
ROP INDEX IF EXISTS Products.PhoneIndex;
```

7.8. Ejercicio práctico: instalación con Docker

En este último apartado se explica cómo realizar la instalación de Apache Cassandra con Docker sobre Windows.

Las imágenes de Docker están prediseñadas y configuradas para facilitar su uso y manipulación (Shirinbab et al., 2019). No es necesario crear máquinas virtuales completas para testear distintos productos en nuestros ordenadores.

Actualmente todas las bases de datos NoSQL tienen una imagen prediseñada en Docker, con lo cual, el trabajo que hagas ahora en esta práctica te será útil para que puedas usar dichas bases de datos, sin instalaciones complicadas sobre tu ordenador.

¡Manos a la obra!

- ▶ **Paso 1:** instala Docker Desktop sobre tu sistema operativo. No es necesario crear una cuenta en Docker Hub para usar los servicios de Docker Desktop.

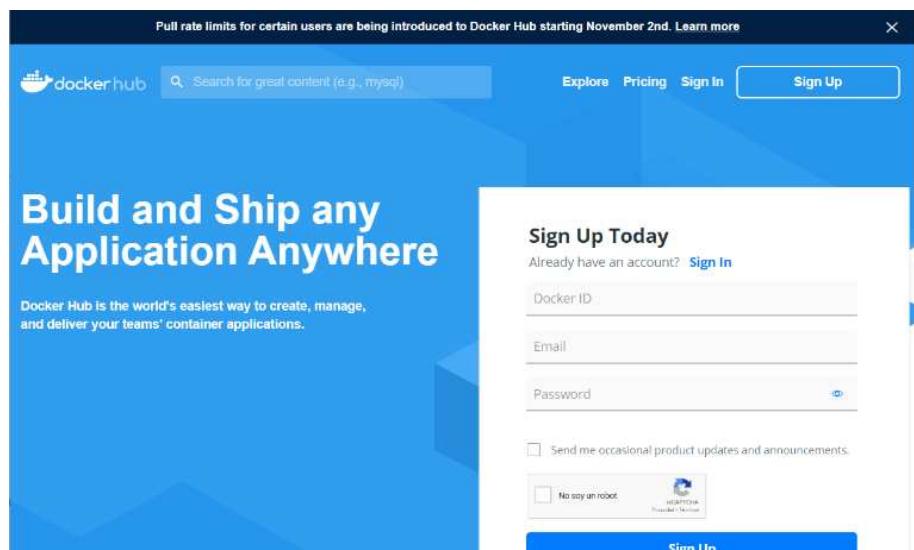


Figura 5. Web de inicio de Docker Hub.

- ▶ **Paso 2:** en el siguiente enlace tienes la información necesaria para llevar a cabo dicha instalación <https://docs.docker.com/docker-for-windows/install/>. Se asume que esta actividad se hará sobre Windows.



Figura 6. Web de Docker Hub, instrucciones de instalación de su herramienta.

- ▶ **Paso 3:** inicia la aplicación de Docker Desktop y luego abre una terminal de CMD o PowerShell en Windows y ejecuta la instrucción:

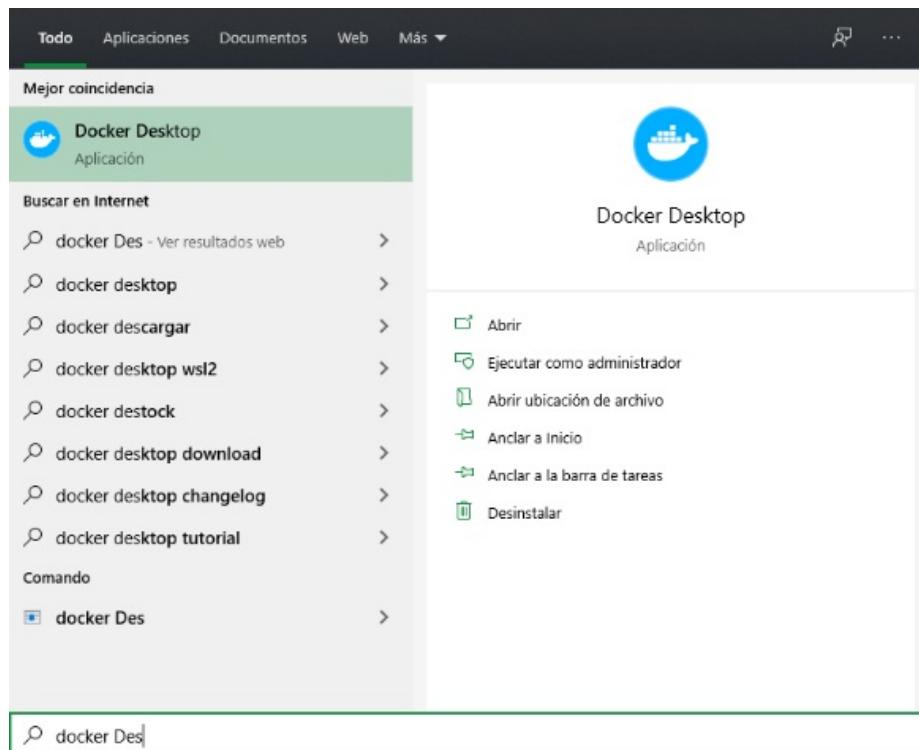
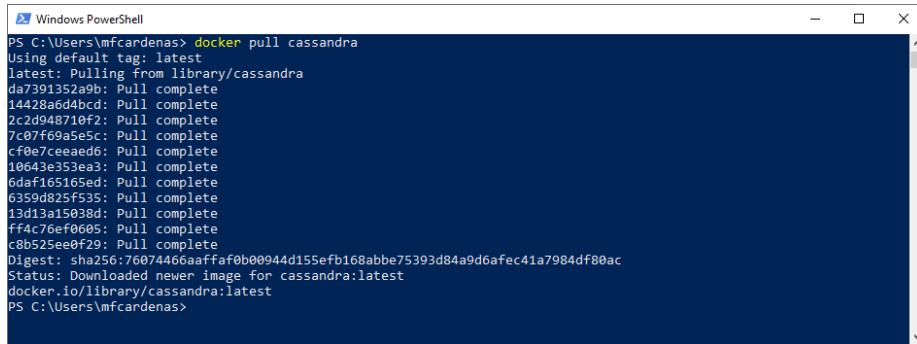


Figura 7. Menú inicio con la opción Docker Hub una vez instalado.

- ▶ **Paso 4:** desde PowerShell inicia la descarga de la imagen de Cassandra con la siguiente instrucción:
 - docker pull Cassandra



```
PS C:\Users\mfcardenas> docker pull cassandra
Using default tag: latest
latest: Pulling from library/cassandra
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
7c07ff69a5e5c: Pull complete
cf0e7ceeaed6: Pull complete
10643e3353ea3: Pull complete
6daf165165ed: Pull complete
6359d825f535: Pull complete
13d13a15038d: Pull complete
ff4c76ef0605: Pull complete
c8b525ee0f29: Pull complete
Digest: sha256:76074466aaffaf0b00944d155efb168abbe75393d84a9d6afec41a7984df80ac
Status: Downloaded newer image for cassandra:latest
docker.io/library/cassandra:latest
PS C:\Users\mfcardenas>
```

Figura 8. Ejecución de instalación de imagen Cassandra vía Docker.

Como puedes observar en la imagen, se ha descargado en tu ordenador partes o fragmentos de una máquina virtual que Docker gestionará por ti. Tú simplemente tendrás que hacer uso de ella y divertirte jugando con Cassandra.

- ▶ **Paso 5:** en Docker Desktop visualiza la nueva imagen que has descargado.

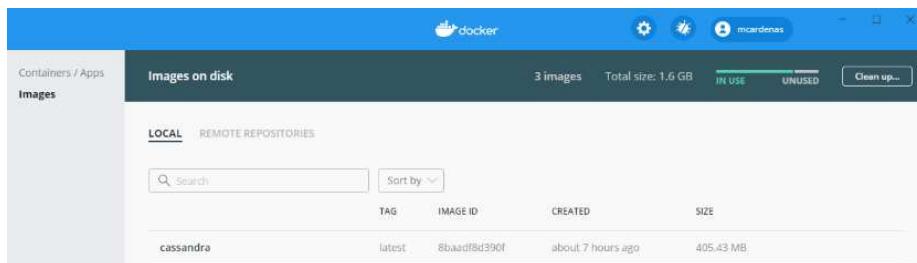


Figura 9. Visualización de la imagen de Cassandra una vez ha sido creada con Docker Hub.

Si posicionas el cursor sobre la imagen, podrás acceder a varios botones que te permitirán manipular la imagen. Ahora solo interesa que ejecutes (*RUN*) dicha imagen.

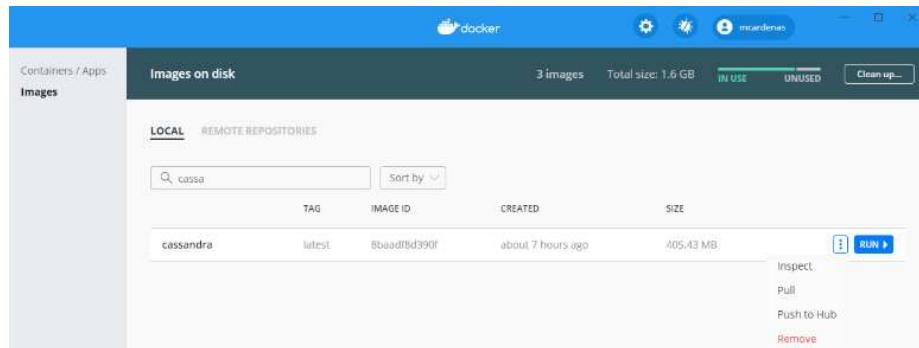


Figura 10. Opciones disponibles sobre la imagen de Cassandra en Docker Hub.

Al ejecutar la imagen, te pedirá una información que será útil para dos cosas:

- ▶ Diferenciar el puerto de acceso a Cassandra desde tu PC del puerto que internamente use la imagen.
- ▶ Establecer una ruta para que los datos (base de datos de Cassandra) se aloje en un directorio de vuestro PC.

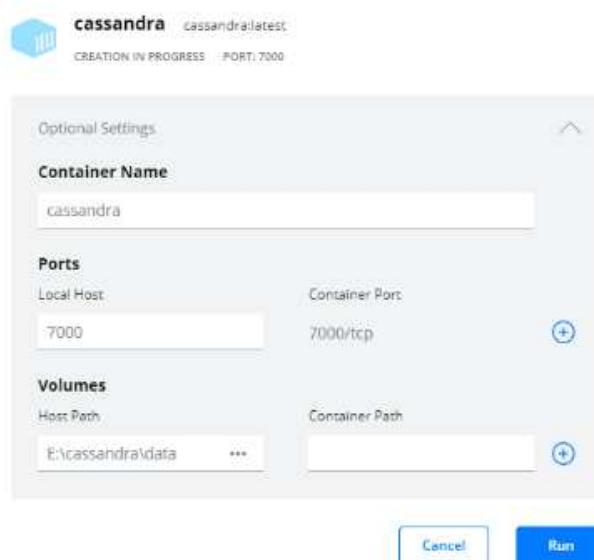


Figura 11. Configuración de la imagen de Cassandra en Docker Hub.

Si das clic en el contenedor Cassandra creado, podrás ver la salida de *log* que ha resultado de ejecutar o levantar la imagen.

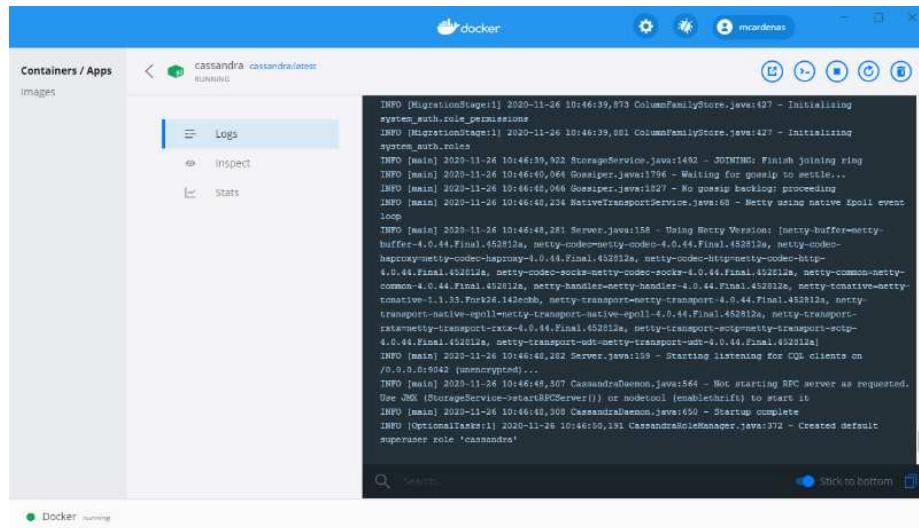


Figura 12. Ventana de Log de la imagen de Cassandra en Docker Hub.

- ▶ **Paso 6:** accede a la consola `cqlsh` desde la propia imagen, es decir, accediendo a la línea de comandos de la imagen y luego a `cqlsh`.
- `docker exec -it cassandra bash` .
- Aparecerá esto: `root@xXxX:/#` . Es la línea de comando de tu imagen de Cassandra.
- Sobre la línea de comando de la imagen, ejecuta `cqlsh` .

The screenshot shows a terminal window with the command `docker exec -it cassandra bash` run in it. Once inside the container, the user runs `cqlsh`. The terminal output shows the connection to the 'Test Cluster' at port 9042, and the version information: [cqlsh 5.0.1 | Cassandra 3.11.9 | CQL spec 3.4.4 | Native protocol v4]. It also indicates that help can be accessed via the `USE HELP` command.

```

PS C:\Users\mfcardenas> docker exec -it cassandra bash
root@1c21fbfc76b5:/# cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.9 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>

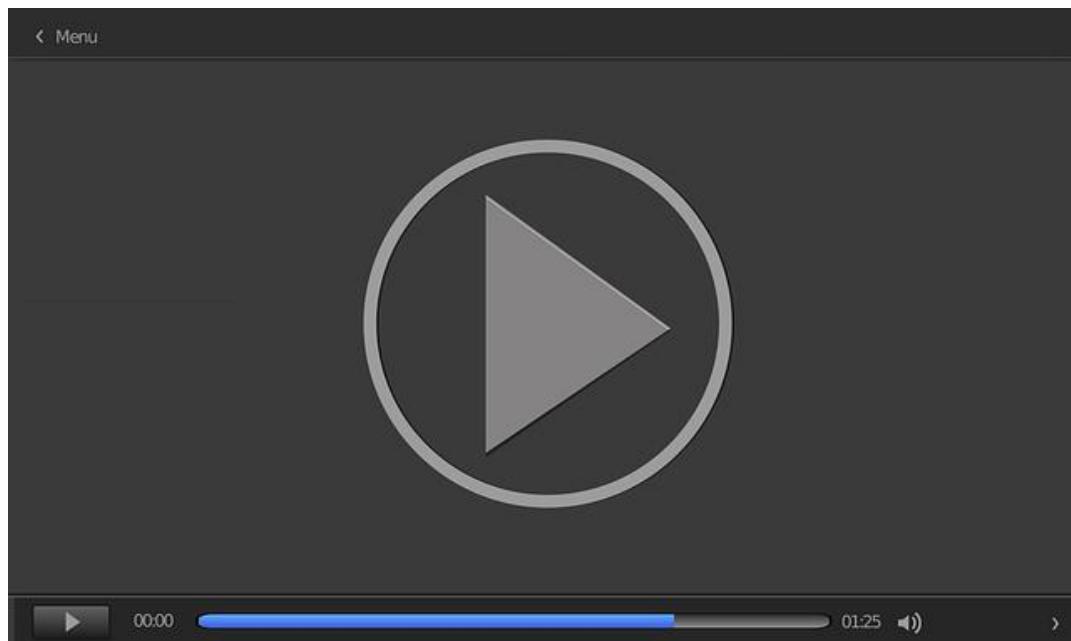
```

Figura 13. Consola Bash de la imagen de Cassandra en Docker Hub.

¡Enhorabuena! En este punto dispones de imagen Docker de Cassandra para que

puedas hacer tus pruebas. Úsala para las distintas actividades prácticas que el profesor proponga en las clases.

En el vídeo «Características de Cassandra» se hace una introducción al uso de Cassandra y sus particularidades.



Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=07ebd201-ee2a-4821-a0f0-aca80143c110>

Vídeo 1. Características de Cassandra.

7.9. Referencias bibliográficas

Carpenter, J., y Hewitt, E. (2020). *Cassandra. The definitive guide. Distributed data at web scale*. O'Reilly Media.

Kan, C. Y. (2014). *Cassandra data modeling and analysis*. Packt Publishing.

Neeraj, N. (2013). *Mastering Apache Cassandra*. Packt Publishing.

Sharma, S. (2014). *Cassandra design patterns*. Packt Publishing.

Shirinbab, S., Lundberg, L., y Casalicchio, E. (2019). Performance comparision between scaling of virtual machines and containers using Cassandra NoSQL database. En B. Duncan, Y. Woo Lee, M. Westerlund, y A. Aßmuth (eds.), *Cloud computing* (pp. 93-98). IARIA.

La documentación oficial de Apache Cassandra

The Apache Software Foundation. (2016). Apache Cassandra Documentation v4.0-beta5. Apache Cassandra [Archivo de vídeo]. <https://cassandra.apache.org/doc/latest/>

Esta página recoge toda la documentación de las distintas distribuciones de Apache Cassandra.



Introducción a Cassandra con Datastax

DataStax Developers. (10 de abril de 2019). *Introduction to Apache Cassandra™ + What's New in 4.0 by Patrick McFadin | DataStax Presents* [Archivo de vídeo]. <https://www.youtube.com/watch?v=d7o6a75sfY0>



Este vídeo muestra una introducción de Apache Cassandra como parte del equipo de Datastax. Describen su funcionalidad, sus principales componentes y la arquitectura que caracteriza a esta base de datos.

Webinar sobre el modelado de datos en Cassandra

DataStax. (9 de marzo de 2020). *How to Create a Cassandra Data Model [Webinar]* / DataStax [Archivo de vídeo]. <https://www.youtube.com/watch?v=4D39wJu5Too>



Create a Cassandra Data Model

WEBINAR

Patrick McFarlin & Jeff Carpenter | DataStax Developer Relations



Esta es, sin duda alguna, una de las mayores cuestiones que surgen alrededor del uso de Apache Cassandra como motor de base en una solución final. En este webinar el equipo de DataStax brinda un conjunto de sugerencias útiles al momento de definir un modelo de datos sobre Cassandra.

Instalar Cassandra en Windows

Kaplarevic, V. (2020). How to Install Cassandra on Windows 10. *phoenixNAP* [Página web]. <https://phoenixnap.com/kb/install-cassandra-on-windows>

La instalación de Cassandra en Windows es posible y no siempre está bien documentada. En este enlace explican con mayor claridad cómo instalar Apache Cassandra con el zip propio del sitio oficial de Apache. La sugerencia es que instales la versión propia de Apache y aprendas a realizar una instalación «pura».

Bibliografía

Matallah, H., Belalem, G., & Bouamrane, K. (2020). Evaluation of NoSQL databases: MongoDB, Cassandra, HBase, Redis, Couchbase, OrientDB. *International Journal of Software Science and Computational Intelligence (IJSSCI)*, 12(4), 71-91.

1. Cassandra:

- A. Es un sistema de almacenamiento de datos NoSQL desarrollado por Facebook.
- B. Es un sistema de almacenamiento en tiempo real para aplicaciones en línea.
- C. Está diseñado para manejar cargas de trabajo en múltiples nodos.
- D. Todas las afirmaciones anteriores son correctas.

2. ¿Cuáles son los componentes principales de Cassandra?

- A. *Cluster, keyspace, column y column & family.*
- B. Columna *name, tables* y *keyspace*.
- C. Mem-Table, SSTable y Bloom Filter.
- D. Las respuestas A y C son correctas.

3. ¿Cuáles son las colecciones en CQL Cassandra?

- A. *Tupple, list y timestamp.*
- B. *Map, list* y *set*.
- C. *Counter, duration* y *date*.
- D. Ninguna respuesta anterior es correcta.

4. ¿Cuál es la principal característica de la colección *list*?

- A. Almacenar datos de forma aleatoria.
- B. Almacenar datos de forma ordenada y que se puedan repetir.
- C. Almacenar elementos clave-valor.
- D. Almacenar elementos para usarlos en un orden concreto.

5. ¿Qué pasa con los datos eliminados en Cassandra?
 - A. Se borran inmediatamente de la base de datos.
 - B. Se almacenan temporalmente en la papelera de reciclaje de Cassandra.
 - C. Son marcados con una lápida.
 - D. Nunca se borran, están inactivos hasta que el usuario los vuelva a activar.
6. ¿Cuál de las siguientes afirmaciones es verdadera?
 - A. Cassandra escribe los datos en una caché clave-valor llamada Mem-Table.
 - B. Los datos en Mem-Table se ordenan por clave.
 - C. Existe una Mem-Table por cada ColumnFamily y de ella se recuperan los datos por la columna clave.
 - D. Todas las afirmaciones anteriores son correctas.
7. La instrucción ALTER KEYSPACE se puede utilizar para:
 - A. Definir un esquema.
 - B. Crear una tabla.
 - C. Ejecutar una consulta.
 - D. Modificar un *keyspace*.
8. Al intentar borrar un elemento que no existe en una colección *set*, se produce:
 - A. La inserción de un nuevo elemento con dicho valor.
 - B. Un error en la operación de borrado.
 - C. Una operación que no se lleva a cabo y que tampoco genera error alguno.
 - D. Una mutación de la colección.

9. ¿Qué es una UDT?

- A. Un tipo de datos primitivo.
- B. Un objeto con funciones especiales en Cassandra.
- C. Un tipo de datos definido por el usuario.
- D. Un proceso de carga de datos.

10. ¿Cuál de las siguientes instrucciones es correcta?

- A. select * from client.agent where dept='AB';
- B. drop index IF EXISTS clients.DeptIndex;
- C. insert into University.Teacher(id,Name,Email,Description) values (2, 'Hamilton', {'hamilton@hotmail.com'}, ['Data Science']);
- D. Todas las instrucciones anteriores son correctas.

Bases de Datos para el Big Data

Neo4j

Índice

Esquema	3
Ideas clave	4
8.1. Introducción y objetivos	4
8.2. Descarga e instalación	5
8.3. Conceptos generales	10
8.4. Graph Data Modelling	15
8.5. Interfaz Neo4j	17
8.6. CQL	19
8.7. Visualización de grafos	29
8.8. Caso práctico	33
8.9. Referencias bibliográficas	40
A fondo	41
Test	44

Esquema

Neo4j	
Definición	Principales componentes
<p>Neo4j es una base de datos de grafos cuyo lenguaje de consultas es CQL (Cypher Query Language). Neo4j está escrito en Java.</p>	<p>Nodo:</p> <ul style="list-style-type: none"> - Las entidades reciben el nombre de nodos y cada uno de ellos es muy similar a una instancia de un objeto (es decir, tienen propiedades). <p>Relaciones:</p> <ul style="list-style-type: none"> - Las relaciones, por su parte, se conocen como vértices y también tienen propiedades, siendo la dirección o sentido el más importante de ellos. Una relación conecta dos nodos y los organiza en estructuras. <p>Base de datos NoSQL de tipo grafo</p> <p>Cómo aplica el teorema CAP</p>
<p>Los principales elementos del modelo de datos Graph DB son:</p> <ul style="list-style-type: none"> - Nodos. - Relaciones. - Propiedades. 	<p>Propiedades:</p> <ul style="list-style-type: none"> - Nodo y relaciones permiten organizar los datos de tal manera que sea posible encontrar patrones de información existente entre dichos nodos. - Las propiedades son pares de nombre-valor utilizados para agregar cualidades a los nodos y las relaciones. <p>Manipulación del grafo</p> <ul style="list-style-type: none"> • Tipos de datos y operadores. • Nodos. • Relaciones. • Propiedades. • Modelado. <p>neo4j</p> <p>Relación (Vértice)</p>
	<p>Caso de uso: fraude en tarjetas de crédito</p>

8.1. Introducción y objetivos

Este tema pretende ser una guía para empezar a desarrollar aplicaciones utilizando Neo4j. Para estudiar este tema debes leer las Ideas clave y, si deseas información adicional sobre un concepto específico, puedes consultar las lecturas citadas en «Referencias bibliográficas». Se recomienda también intentar replicar los ejemplos para entender cada uno de los conceptos que se explican.

Para poder darle un enfoque práctico a este tema, es recomendable que el estudiante instale su base de datos de prueba siguiendo las instrucciones del apartado 8.2. Una vez instalado el entorno, continúa con los epígrafes siguientes poniendo en práctica todos los comandos citados de Neo4j.

Una de las bases de datos de grafos más conocidas es Neo4j. Es de código abierto y ha sido desarrollada utilizando Java. Sus principales características consisten en que es escalable en gran medida y no presenta esquema (es NoSQL).

Este tipo de base de datos se basa en la representación gráfica de un conjunto de objetos donde algunos pares de ellos están conectados por enlaces. Hay dos elementos claves en Neo4j: los nodos (vértices) y las relaciones (bordes).

Con base en estos dos elementos, Neo4j se utiliza para modelar los datos en forma de grafo con relaciones. En el grafo los nodos representan las entidades, mientras que las relaciones indican la asociación que hay entre los nodos.

Neo4j no es la única base de datos de este tipo, existen otras que siguen esta misma topología, entre ellas OrientDB, GraphBase, Oracle NoSQL Database, HyperGraphDB, InfiniteGraph y AllegroGraph.

Los objetivos que se pretenden conseguir con este tema son los siguientes:

- ▶ Comprender las ideas claves que describen el concepto de grafo.
- ▶ Conocer la utilidad que tienen los grafos como forma de representar los datos y sus relaciones.
- ▶ Aplicar los conceptos de grafo sobre la base de datos Neo4j.
- ▶ Aprender a instalar y configurar un entorno de pruebas local con Neo4j.

8.2. Descarga e instalación

Antes de hablar de la instalación de Neo4j, es interesante que conozcas las versiones que existen y se pueden descargar para trabajar con esta base de datos. Al igual que MongoDB, Neo4j cuenta con una versión Enterprise y con una versión Community.

La versión Enterprise ofrece más funcionalidades propias de un sistema en producción, mientras que la Community está más orientada a entornos de desarrollo. De cara a este tema, es recomendable trabajar sobre la versión Community.

Otro aspecto importante es que Neo4j necesita que tengas Java en el ordenador, y esto puede ser algo nuevo si no conoces cómo trabaja Java. Aunque puedes encontrar imágenes de Docker con Neo4j listas para usar, con el fin de que conozcas cómo es una instalación propia de esta herramienta, se propone que hagas dicha instalación sobre tu ordenador siguiendo los pasos que se describen a continuación.

Además de instalar Neo4j, también conocerás el concepto de variables de entorno en Windows, las cuales son útiles para configurar herramientas que utilizan este tipo de variables (es el caso de Java y, por ende, de Neo4j).

- **Paso 1:** descarga la versión Community de Neo4j.

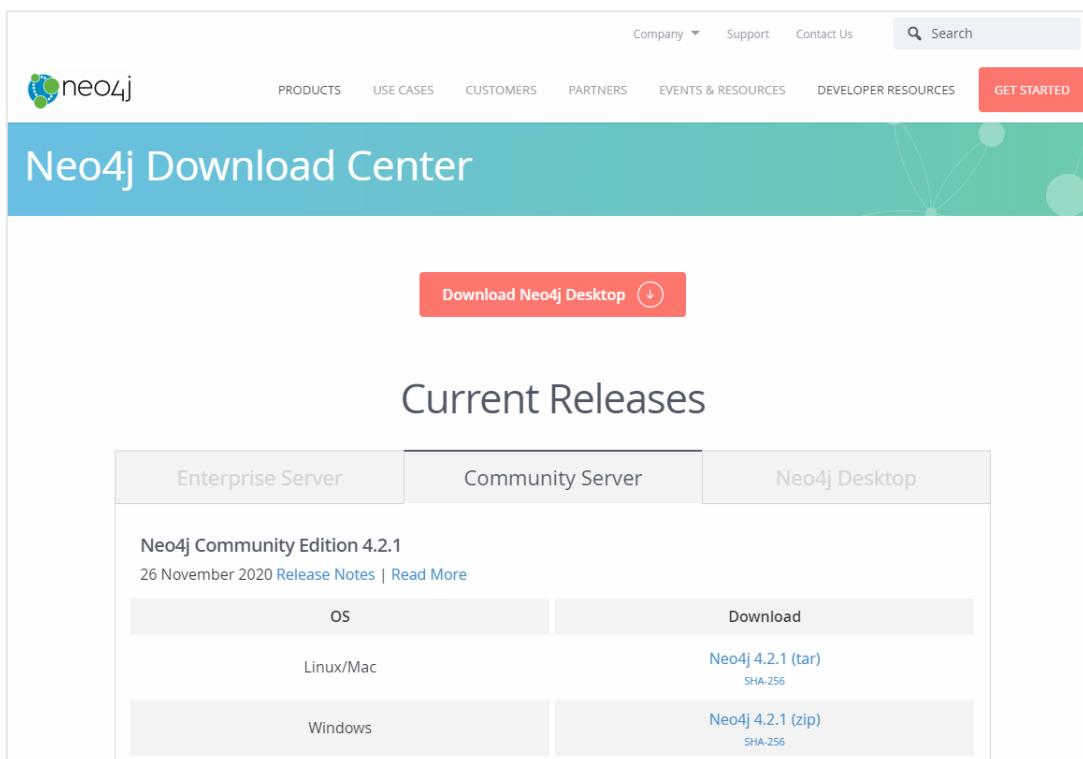


Figura 1. Web de descarga de MongoDB.

- **Paso 2:** si no tienes instalado Java en tu ordenador, descarga la versión *OpenJDK 8* u *Oracle Java 8*. Esta versión es la recomendada para Neo4j 3.0.x. Si por alguna razón debes usar una versión de Neo4j inferior a 2.3.0, es recomendable usar la versión 7 de Java. Consulta el apartado «Instalar Java 8 y crear variable de entorno» de este documento.
- **Paso 3:** el archivo que has descargado en el Paso 1 es un *zip* que contiene los ficheros de Neo4j. Localiza dicho fichero y extrae su contenido dentro de un directorio conocido, por ejemplo: **D:\Neo4j**.
- **Paso 4:** el ordenador donde se instala Neo4j hará las veces de servidor. Es recomendable crear la variable de entorno NEO4J_HOME para señalar el directorio del Paso 3. En dicho servidor, es posible ejecutar una instancia de la consola de Neo4j para trabajar (Paso 5.1), pero también es posible crear un servicio en Windows que permita levantar la base de datos de forma permanente,

igual que Mongo (Paso 5.2). Si piensas utilizar Neo4j de forma permanente, se recomienda instalar el servicio.

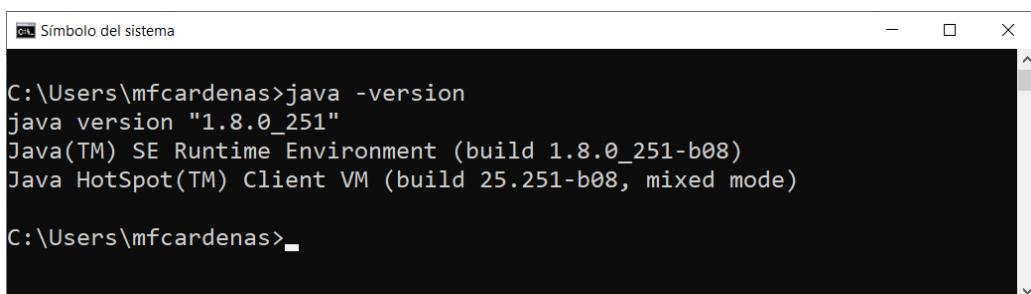
- ▶ **Paso 5.1:** para ejecutar una instancia de Neo4j, desde la consola de Windows PowerShell, ubícate en la ruta y ejecuta `neo4j console` (`<NEO4J_HOME>\bin\neo4j console`). Ten en cuenta que al descomprimir el fichero ZIP, este crea otro directorio llamando `/neo4j-community-4.x.x`. Se asume que su contenido es lo que has copiado en el directorio recomendado en el Paso 3. Si lo que copias es el directorio completo, entonces ten en cuenta este nuevo directorio en la ruta propuesta.
- ▶ **Paso 5.2:** para instalar Neo4j como servicio, utiliza la instrucción `<NEO4J_HOME>\bin\neo4j install-service`.
- ▶ **Paso 6:** si todo ha ido bien, accede a la URL <http://localhost:7474> desde el navegador web que usas frecuentemente. Utiliza el usuario `neo4j` con la contraseña predeterminada `neo4j`. De inmediato te pedirá que cambies la contraseña para mayor seguridad.

Observa cómo cada base de datos utiliza un puerto por defecto en las instalaciones por defecto: MySQL utiliza el puerto 3306, MongoDB el puerto 27017, Neo4j el puerto 7474... Lo más probable y seguro es que los administradores de bases de datos asignen puertos menos comunes.

Instalar Java y crear variable de entorno

- ▶ **Paso 1:** descarga Java 8 o superior del sitio oficial de Oracle (intenta en la siguiente URL: <https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>). Si falla la URL anterior (es Oracle y cada vez que innovan mueven cosas de su sitio web), busca en Google «Instalar Java 8».

- ▶ **Paso 2:** ejecuta el ejecutable que has descargado e instala Java en la ruta D:\Java\jdk1.8.0 (D es una unidad de ejemplo, puedes usar otra que consideres más oportuna).
- ▶ **Paso 3:** desde Windows crea la variable JAVA_HOME, para ello da clic derecho en Equipo o Este Equipo en el Explorador de archivos de Windows.
- ▶ **Paso 4:** selecciona la opción Propiedades y luego accede a Configuración avanzada del sistema > Variables de entorno.
- ▶ **Paso 5:** en el recuadro Variables del sistema crea una «Nueva...» variable del sistema cuyo nombre debe ser JAVA_HOME. A dicha variable ponle como valor de la variable la ruta de instalación de Java, es decir, D:\Java\jdk1.8.0. Acepta todas y cada de las ventanas abiertas (son unas cuantas).
- ▶ **Paso 6:** comprueba que todo está correcto. Para ello, abre una consola de Windows y escribe las instrucciones SET JAVA_HOME para comprobar la ruta y luego java -version para mostrar la versión que acabas de instalar de Java.



```
C:\Users\mfcardenas>java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) Client VM (build 25.251-b08, mixed mode)

C:\Users\mfcardenas>
```

Figura 2. Visualizar la versión de Java desde la consola de Windows.

```

Windows PowerShell
PS E:\Neo4J\bin> dir
Directorio: E:\Neo4J\bin

Mode                LastWriteTime     Length Name
----                -----        -- Neo4j-Management
d----

```

Figura 3. Modelo de referencia arquitectura Apache Cassandra.

Si todo ha ido bien, podrás ver la siguiente pantalla en la URL <http://localhost:7474>.

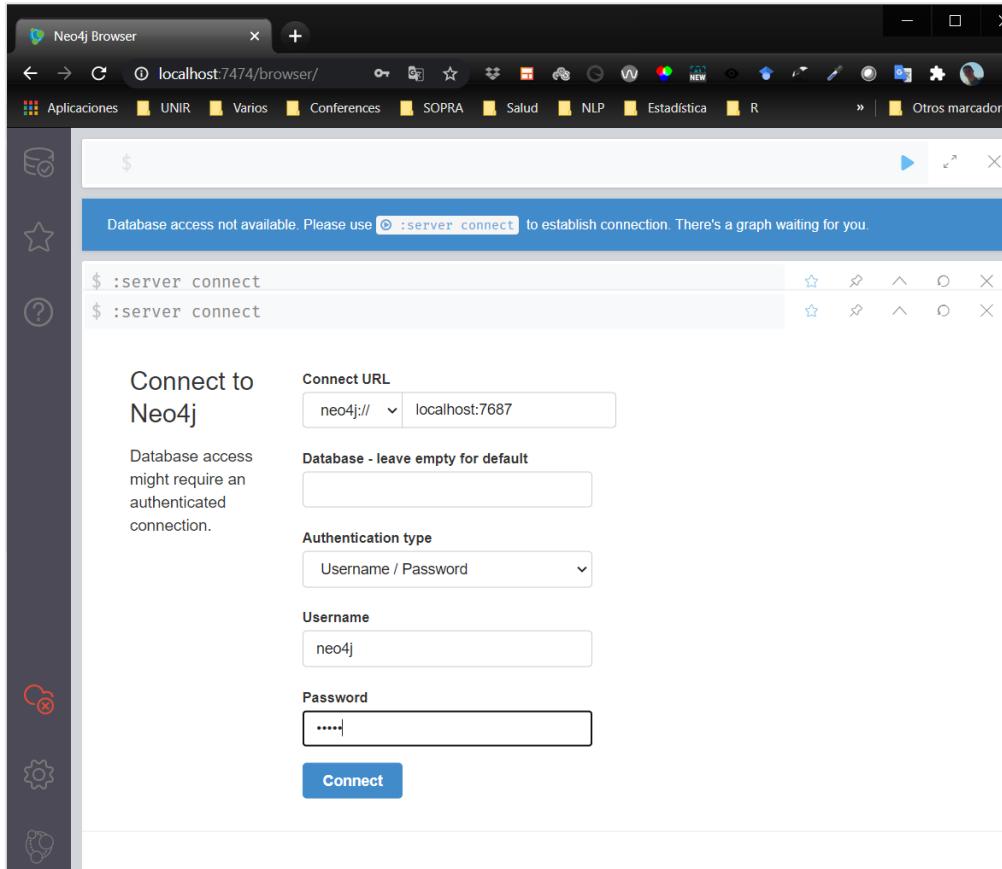
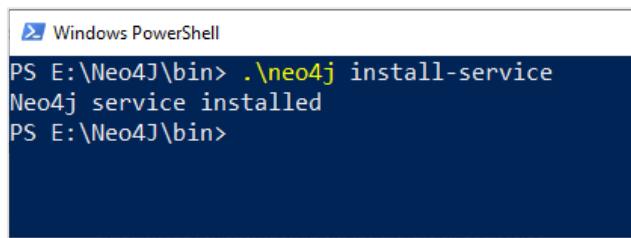


Figura 4. Pantalla de configuración de Neo4j.

Instalar Neo4j como servicio en Windows

Para instalar Neo4j como servicio, basta con seguir dos pasos:

- ▶ **Paso 1:** desde la ruta de instalación ejecuta lo siguiente: <NEO4J_HOME>\bin\neo4j install-service.



```
Windows PowerShell
PS E:\Neo4J\bin> .\neo4j install-service
Neo4j service installed
PS E:\Neo4J\bin>
```

Figura 5. Instalar servicio de Neo4j desde la consola de Windows.

- ▶ **Paso 2:** comprueba que el servicio está funcionando.

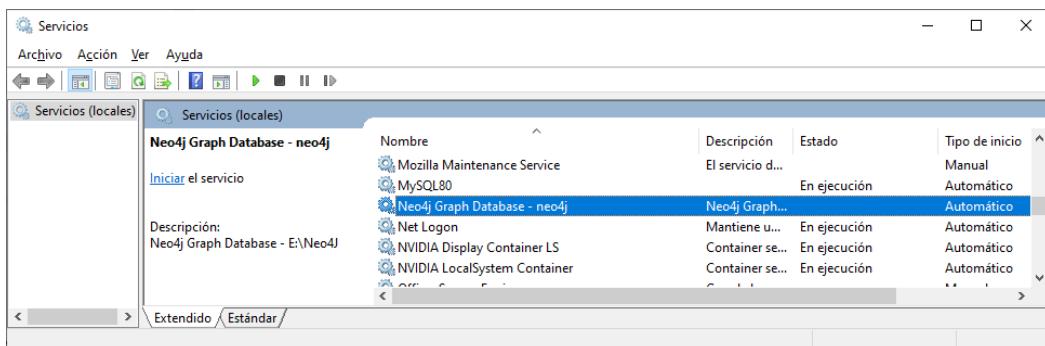


Figura 6. Lista de servicios en Windows.

8.3. Conceptos generales

Los grafos son representaciones gráficas de entidades y sus relaciones. Los nodos representan esas entidades y poseen propiedades que los describen (por ejemplo, el nombre) (Lal, 2015). En el grafo de la imagen siguiente, cada nodo tiene un nombre y está relacionado con otro a través de una flecha que muestra dicha relación (Hölsch et al., 2017).

Las relaciones también tienen múltiples propiedades, las cuales permiten organizar la información asociada a los nodos (Guia et al., 2017; Webber y Robinson, 2018). Las relaciones tienen una dirección (ver Figura 7), pero si se quisiera indicar bidireccionalidad, es necesario crear dos relaciones en sentidos opuestos.

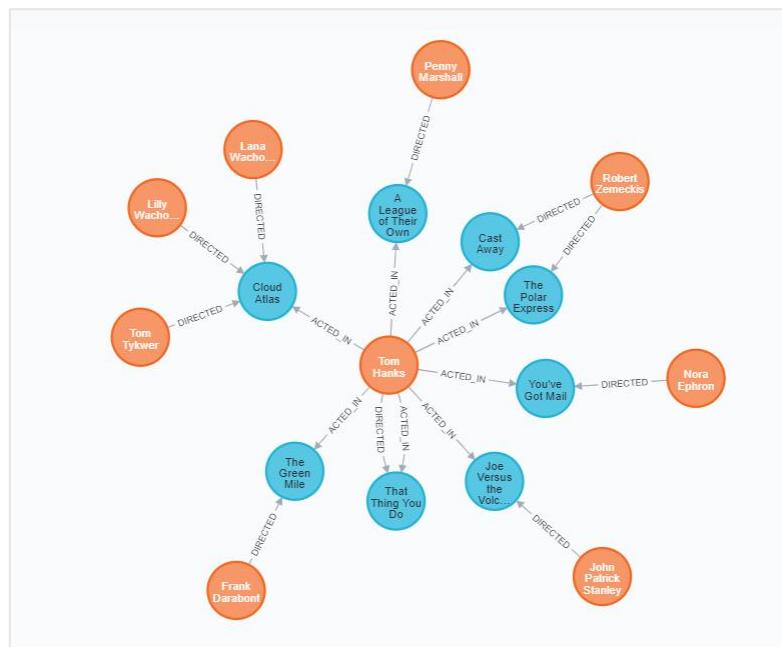


Figura 7. Representación gráfica de los elementos de Neo4j.

Nodos

Los grafos representan entidades y las relaciones entre ellas. Las entidades reciben el nombre de nodos y cada nodo es muy similar a una instancia de un objeto (es decir, tiene propiedades).

Relaciones

Las relaciones por su parte se conocen como vértices y también tienen propiedades, siendo la dirección o sentido el más importante de ellos. Una relación conecta dos nodos y los organiza en estructuras.

De esta forma, un grafo puede mostrarse como una lista, un árbol, un mapa o una entidad compuesta, con la posibilidad de combinarse en estructuras aún más complejas y ricamente interconectadas (grandes y complejas redes de nodos).

Nodo y relaciones permiten organizar los datos de tal manera que sea posible encontrar patrones de información existente entre dichos nodos (Webber y Robinson, 2018). Una vez almacenados los datos, los grafos permitirán interpretar y generar información de valor de diferentes formas, según las relaciones que los nodos tengan (ver Figura 8).



Figura 8. Representación gráfica de nodo y vértice.

Tipos de relaciones

Las relaciones tienen asociado un tipo de relación. En la base de datos de ejemplo de Neo4j, las relaciones que se identifican son ACTED_IN y DIRECTED y se usan como tipos de relación.

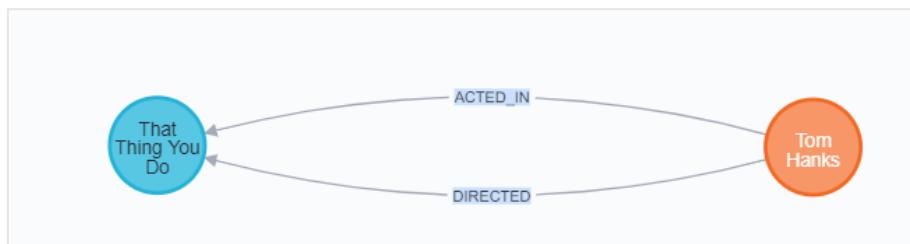


Figura 9. Representación de las relaciones de los nodos.

En la Figura 9 se muestra una relación saliente ACTED_IN (actuó en) con el nodo de «Tom Hanks» como nodo de origen y «That Thing You Do» como nodo de destino. De igual

forma, se puede observar cómo el nodo «Tom Hanks» tiene una relación saliente con el mismo nodo «That Thing You Do» pero con otra connotación diferente: DIRECTED.

Toda relación siempre tendrá una dirección. La dirección que se tome en cuenta será la que sea útil para el análisis que se quiere hacer. Esto significa que no es necesario agregar relaciones duplicadas en la dirección opuesta a menos que sea necesario para describir correctamente un caso de uso. Es importante tener presente que un nodo puede tener relaciones consigo mismo.

Propiedades

Se ha dicho previamente que tanto nodos como relaciones poseen propiedades. Las propiedades son pares de nombre-valor utilizados para agregar cualidades a los nodos y las relaciones. En los grafos de ejemplo (ver imágenes) se han utilizado como propiedad el nombre de las relaciones, el nombre del personaje en los nodos, el título de las películas y el año, entre muchos otros.

El valor de la propiedad puede contener diferentes tipos de datos tales como números, cadenas y booleanos (ver Figura 10).

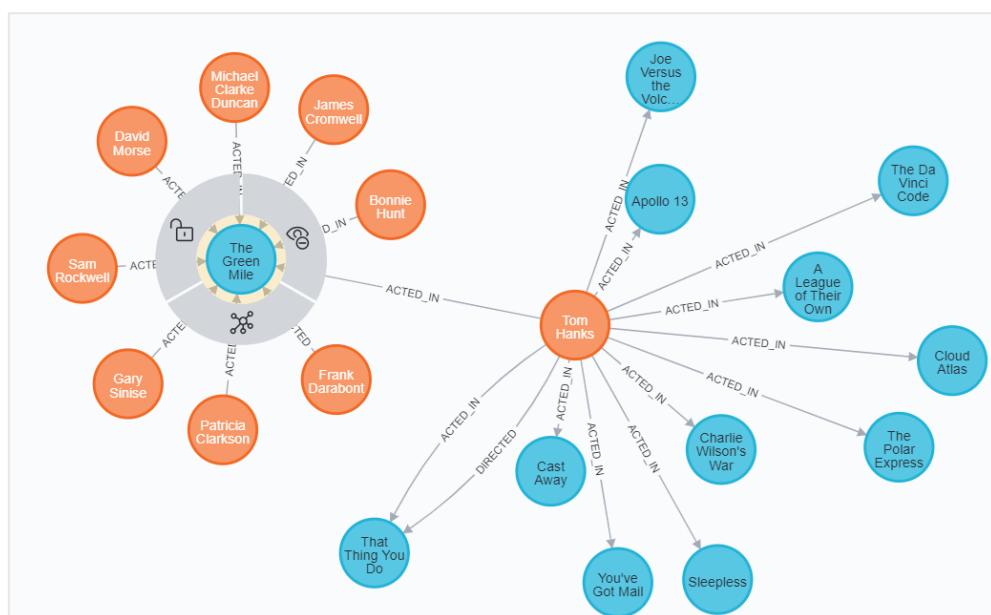


Figura 10. Relaciones entre nodos y su visualización.

Recorridos, rutas o caminos

Recorrer los nodos en función de sus relaciones es lo que se conoce como recorrido. Un recorrido es similar a una consulta del grafo que busca encontrar alguna respuesta a preguntas como: «¿Qué películas ha dirigido Tom Hanks?», «¿En qué películas Tom Hanks, además de actuar, también ha dirigido?», «¿Qué otros actores han trabajado con Tom Hanks en la película *The Green Mile*?».

Cuando se visitan los nodos siguiendo las relaciones y aplicando determinadas reglas, realmente se atraviesa el grafo. Por lo general, no se atraviesa todo el grafo, sino un subconjunto de él.

Si se quisiera saber en qué películas actuó Tom Hanks, si se observa la imagen anterior, el recorrido comenzaría desde el nodo «Tom Hanks», seguiría cualquier relación ACTED_IN conectada al nodo y terminaría con *Forrest Gump* o en todas aquellas películas donde el actor haya actuado.

Esquema

En Neo4j el esquema hace referencia a los índices y restricciones que componen la base de datos. En Neo4j el esquema es opcional, no es necesario crear índices y restricciones sobre la base de datos creada. Incluso es posible crear nodos, relaciones y propiedades sin definir previamente un esquema. Los índices y las restricciones se pueden añadir posteriormente para optimizar el rendimiento de la base de datos y, por ende, del modelo de dato definido.

Índices

Al igual que en los sistemas relacionales, los índices permiten aumentar el rendimiento de las consultas a las bases de datos.

Restricciones

Las restricciones permiten asegurar que los datos cumplan con las reglas del dominio establecido para garantizar coherencia en el modelo.

Recomendaciones de nombres

En lo que respecta a las etiquetas de nodo, los tipos de relación y las propiedades, al igual que Java, son sensibles al uso de mayúsculas y minúsculas. Revisa la tabla siguiente para conocer las principales recomendaciones.

Entidades	Estilos	Ejemplos
Etiqueta del nodo	Comenzar con un carácter en mayúscula.	:PropietarioVehiculo es mejor que :propietario_veiculo
Tipo de relaciones	Todo en mayúsculas usando guion bajo.	:ACTUO_EN es mejor que :actuo_en
Propiedades	Iniciar en minúscula y añadir mayúscula al principio de las siguientes palabras (sin separador).	firstName es mejor que first_name o que firstname

Tabla 1. Recomendaciones para crear elementos en Neo4j.

8.4. Graph Data Modelling

Como se comentó en otros apartados, Neo4j utiliza la representación de grafos para almacenar y administrar sus datos (ver Figura 11). Los aspectos relevantes de cara al modelo de datos son los siguientes:

- ▶ El modelo de datos se representa con nodos (círculos), relaciones (flechas) y propiedades.
- ▶ Tanto los nodos como las relaciones contienen propiedades.

- ▶ Las propiedades también se representan como pares clave-valor.
- ▶ Las relaciones tienen direcciones unidireccionales y bidireccionales.
- ▶ En una relación siempre hay un «nodo de inicio» o «nodo desde» y un «nodo hasta» o «nodo final».
- ▶ Una relación conecta dos nodos.

Cuando se crean relaciones en Neo4j, siempre es necesario indicar una dirección, de lo contrario el motor de base de datos arrojará un mensaje de error diciendo «las relaciones deben ser direccionales». Ejemplo de imagen: si lo que se dice en el texto se puede ilustrar con una imagen, la incluiremos con su correspondiente pie de imagen.

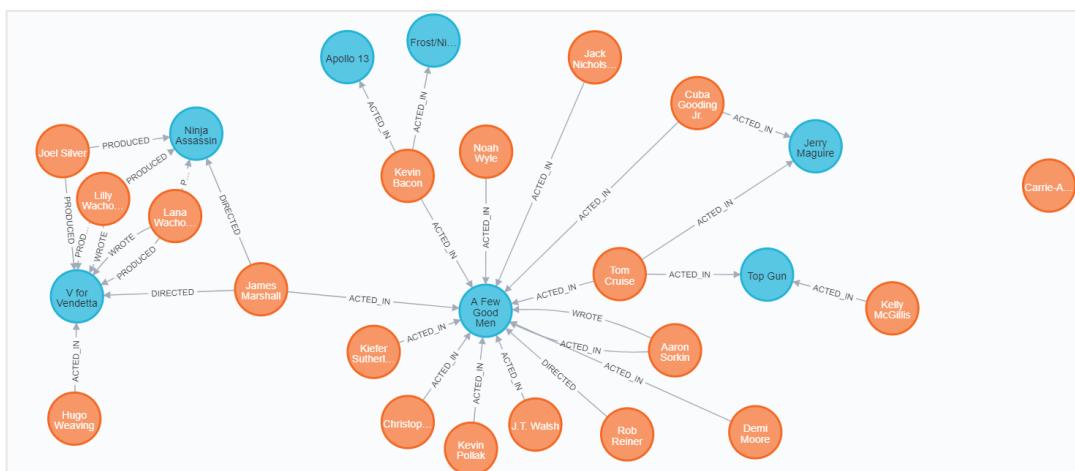


Figura 11. Representación de relaciones mediante grafos.

Al almacenar los datos, no es necesario utilizar otra base de datos SQL o NoSQL para almacenar los datos de la base de datos de Neo4j. Toda la información se guarda en los distintos elementos en su formato nativo. Neo4j utiliza Native GPE (motor de procesamiento de grafos) para trabajar con su formato de almacenamiento de grafos nativo (Fernandes y Bernardino, 2018).

8.5. Interfaz Neo4j

A continuación, se describe la interfaz de la herramienta en su versión 4.2.x. Si observas que no es similar a la que tienes instalada, es porque seguramente estás haciendo pruebas con una versión diferente a la aquí estudiada.

La interfaz de Neo4j es web, y la pantalla de bienvenida se puede observar en la imagen siguiente.

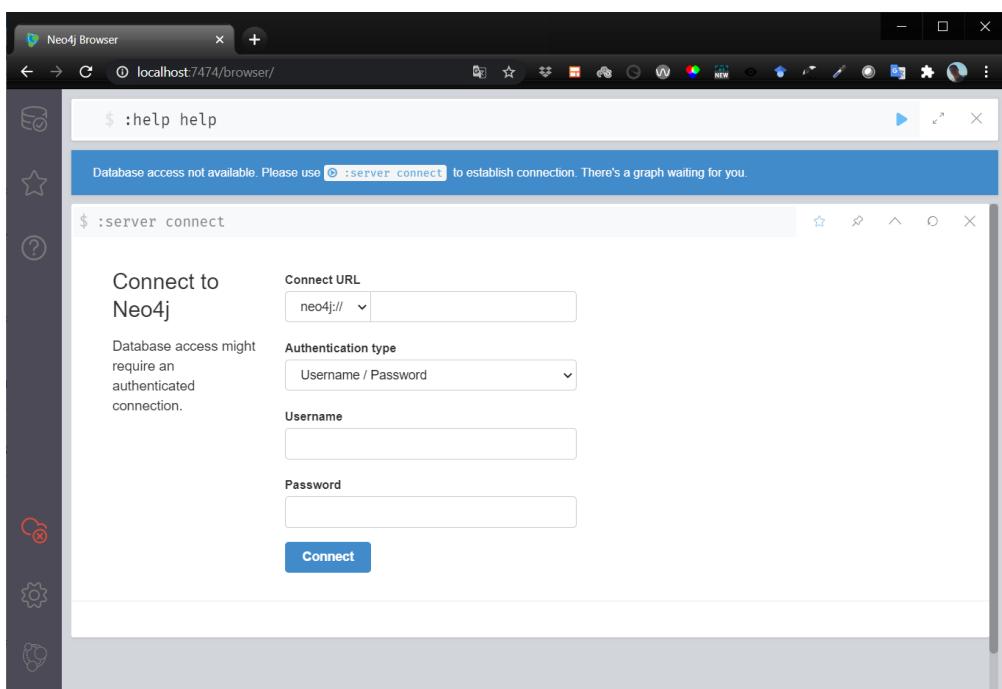


Figura 12. Consola web principal de Neo4j.

A continuación, se describen los principales elementos de la interfaz web.

1. Información de la base de datos: aquí podrás ver toda la información de las bases de datos en Neo4j. Por defecto existen las bases de datos *system* y *neo4j*. Visualiza las etiquetas de los nodos, las relaciones y sus tipos y las propiedades.
2. Favoritos: mostrará las consultas favoritas que hayas marcado como tal, una opción para cargar consultas de ficheros o ejemplos de consultas que podrías utilizar.
3. Ayuda: permite acceder al manual de usuario de Neo4j y Cypher.

4. Opciones de sincronización del buscador.
5. Configuración: permite modificar la apariencia de la web, preferencias de conexión, las características de los datos visualizados y los grafos.
6. Información sobre Neo4j.
7. Línea de comandos principal para ejecutar las consultas con sus respectivas opciones.
Todas las consultas de Neo4j comienzan con dos puntos (:).
8. Línea de comando donde se muestra la consulta ejecutada cuyo resultado se muestra en el área 10.
9. Controles de visualización del resultado de una consulta.
10. Área de visualización del resultado de la consulta.

Cualquier acción que realizamos en Neo4j se traduce en comando o instrucción, es por ello que cuando ejecutes cualquier opción, podrás observar en la línea de comando principal el comando equivalente.

En el área de visualización podrás observar el histórico de todas las operaciones o consultas realizadas, así podrás recuperar la consulta lanzada y reutilizarla siempre que así lo deseas.

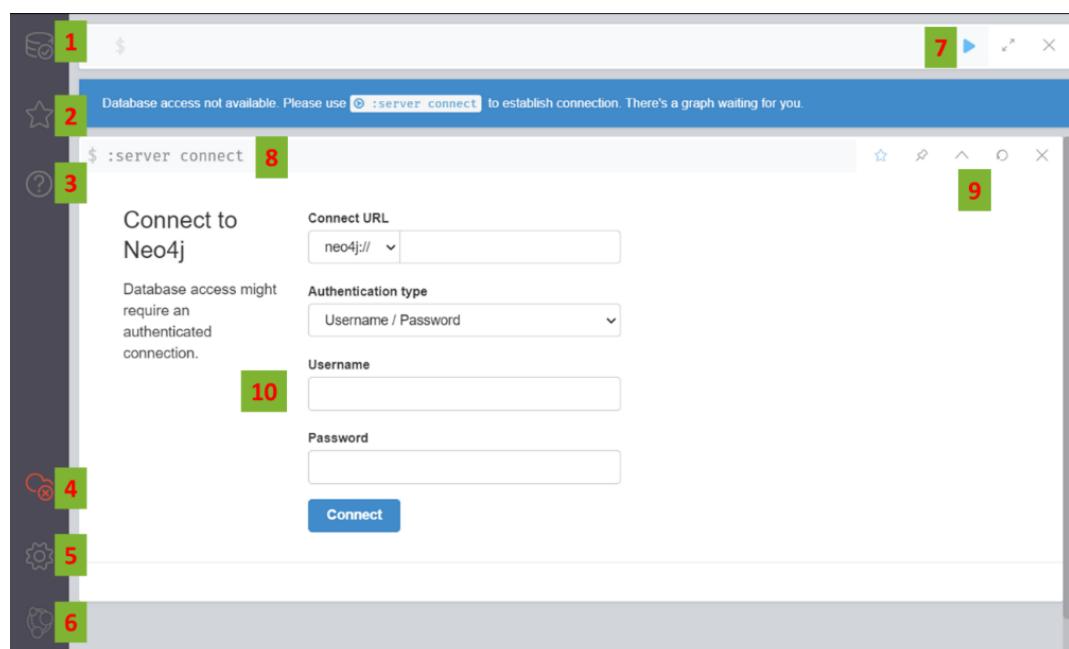


Figura 13. Partes de la consola web de Neo4j.

8.6. CQL

El lenguaje de consulta utilizado en Neo4j es CQL o Cypher Query Language. Es un lenguaje declarativo, que permite buscar coincidencia de patrones. Este sigue una sintaxis similar a SQL, es legible y se puede organizar con determinadas como ocurre con SQL.

CQL emplea comandos o instrucciones para manipular sus bases de datos. Admite muchas cláusulas como WHERE y ORDER BY, entre otras, para escribir consultas complejas de una forma fácil y, digamos, rápida. También permite usar funciones como *string*, *aggregation*, así como algunas funciones de relación.

Tipos de datos

Tipo de dato	Descripción
Boolean	True/False
byte	Enteros de 8 bit
short	Enteros de 16 bit
int	Enteros de 32 bit
long	Enteros de 64 bit
float	Números de 32 bit con punto flotante
doublé	Números de 64 bit con punto flotante
char	Caracteres de 16 bit
string	String

Tabla 2. Tipos de datos en Neo4j.

Operadores

Operador	Descripción
Lógicos	
AND	Operación lógica
OR	Operación lógica
NOT	Operación lógica
XOR	Operación lógica
Comparación	
=	Igualdad
<>	Diferente
<	Menor que
>	Mayor que
<= y >=	Menor igual que, mayor igual que
Otros	
+, -, *, /, %, ^	Matemáticas
+	Uso de <i>string</i>
+, IN, [X], [X.....Y]	Uso de listas
=_	Expresión regular
STARTS WITH, ENDS WITH, CONSTRAINTS	Comparación de <i>string</i>

Tabla 3. Operadores en Neo4j.

Cláusulas de CQL

Cláusula	Descripción
Lectura de datos	
MATCH	Especificar un patrón.
OPTIONAL MATCH	Especificar un patrón donde además es posible usar nulos en caso de que el patrón carezca de partes.
WHERE	Añadir contenido a las consultas CQL.
START	Se utiliza para encontrar los puntos de partida a través de los índices heredados.
LOAD CSV	Se usa para cargar datos desde un fichero CSV.
Escritura de datos	
CREATE	Permite crear nodos, relaciones y propiedades.
MERGE	Comprueba si un patrón especificado existe en el grafo; en caso de que no, crea el patrón.
SET	Permite actualizar etiquetas de nodos, relaciones y propiedades.
DELETE	Permite borrar nodos, relaciones o rutas en el grafo.
CREATE UNIQUE	Mediante las cláusulas CREATE y MATCH, es posible obtener un patrón único al hacer coincidir el patrón existente y crear el que falta.
REMOVE	Permite borrar propiedades de los nodos y relaciones.
FOREACH	Permite actualizar los datos dentro de una lista (recorre la lista).
Otras cláusulas	
LIMIT	Limita el resultado a un número de valores concreto.
SKIP	Se utiliza para establecer desde qué fila comenzar a incluir las filas en la salida.
WITH	Se utiliza para encadenar las partes de la consulta.
RETURN	Se utiliza para definir qué incluir en el conjunto de resultados de la consulta.
ORDER BY	Se utiliza junto con las cláusulas RETURN o WITH y permite organizar la salida de una consulta en orden.
UNWIND	Permite expandir una lista en una secuencia de filas.
UNION	Permite unir el resultado de múltiples consultas.
CALL	Permite llamar a un procedimiento almacenado en la base de datos.

Tabla 4. Cláusulas de CQL en Neo4j.

Funciones

- ▶ **String**: se utiliza para manipular *string*.
- ▶ **Aggregation**: se utiliza en determinadas operaciones de agregación.
- ▶ **Relationship**: se utiliza para obtener información sobre relaciones (nodo inicio, nodo fin, entre otros).

Construir un grafo con CQL

A continuación, utilizarás CQL para crear tu propio grafo y luego ejecutar sobre él determinadas consultas que se irán proponiendo.

InSTRUCCIONES PARA LA CREACIÓN DE NODOS

- ▶ `CREATE (MiNodo)`: un nodo simple.
- ▶ `CREATE (MiNodo1), (MiNodo2)`: múltiples nodos.
- ▶ `CREATE (Andres:Persona)`: crea el nodo `Andres` con la etiqueta `Persona`.
- ▶ `CREATE (Andres:Persona:Padre:Hijo:Prestamista)`: crea el nodo `Andres` con las etiquetas `Persona`, `Padre`, `Hijo` y `Prestamista`.
- ▶ `CREATE (Andres:Persona { name: "Andres Fabricio", age: 21, nationality: "Spain" })`: crea el nodo `Andres` con la etiqueta `Persona` y las propiedades `name`, `age` y `nationality`.
- ▶ `CREATE (Andres:Persona { name: "Andres Fabricio", age: 21, nationality: "Spain" }) RETURN Andres`: después de crear el nodo lo retorna para su visualización.
- ▶ `MATCH (n) RETURN n`: visualiza todos los nodos *n* creados (cuidado: todos los nodos creados).

Ejercicio 1

- ▶ **Paso 1:** ejecuta todas las instrucciones anteriores.

- ▶ **Paso 2:** ejecuta esta otra instrucción:

```
CREATE (Eduardo:Persona{name:"Eduardo Puertas", age:45,  
nacionality:"Español"}), (Camila:Persona{name:"Camila Corre", age:25,  
nacionality:"Frances"})
```

- ▶ **Paso 3:** visualiza todos los nodos.

El resultado debería ser parecido al siguiente:

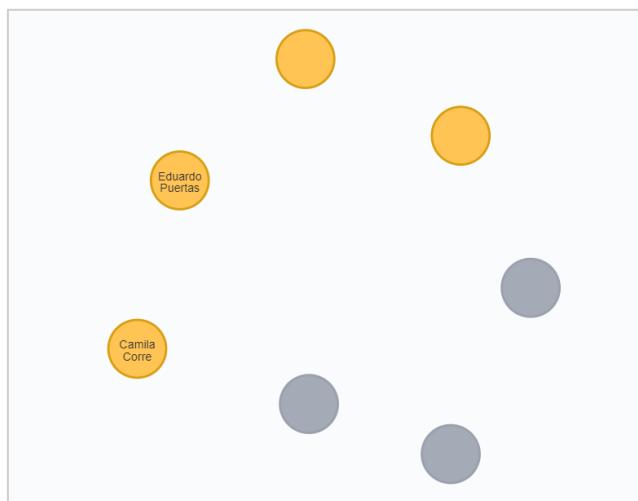


Figura 14. Representación de nodos en Neo4j.

Crear relaciones

- ▶ `CREATE (Ana:Persona{name:"Ana Lana", age:25, nacionality:"Española"}),
(Pepe:Persona{name:"Pepe Rueda", age:25, nacionality:"Francés"})
CREATE (Ana)-[r:AMIGO_DE]->(Pepe)`

Para que la relación se cree utilizando los nombres de los nodos, la creación de estos debe estar en la misma instrucción, es decir, se deben ejecutar los dos CREATE a la vez en la línea principal (no copies, pegues y ejecutes cada instrucción CREATE por separado).

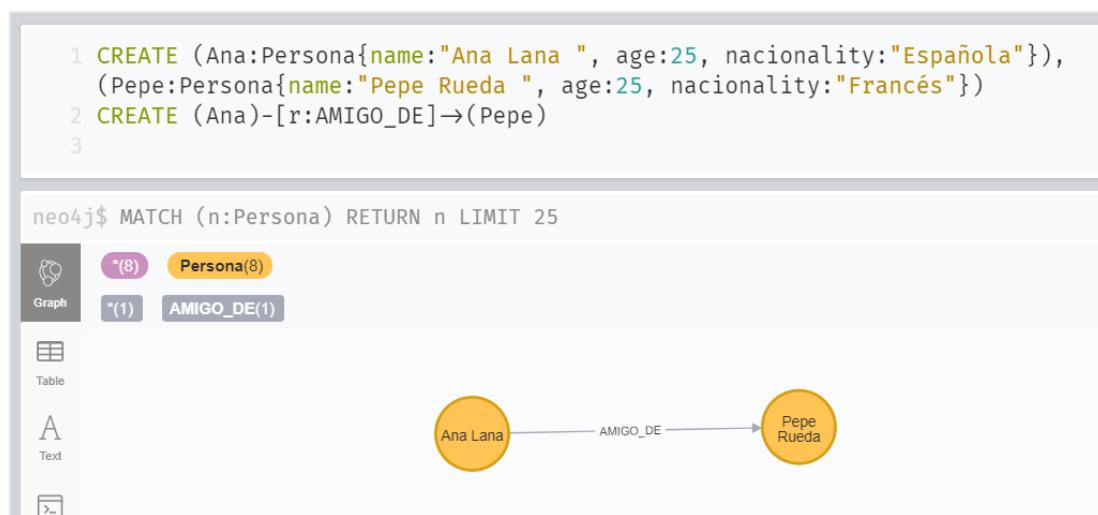


Figura 15. Representación de relación entre nodos en Neo4j.

Otro ejemplo de creación de relación a partir de una MATCH:

```

MATCH (a:Persona), (b:Persona) WHERE a.name = "Eduardo Puertas" AND b.name  

= "Camila Corre"  

CREATE (a)-[r: AMIGO_DE]->(b)  

RETURN a,b
  
```

Observa que, en este caso, a y b representan los nombres (que no se conocen) de los nodos. Lo que sí se conoce de ellos son las etiquetas (:Persona). En el WHERE lo que se hará es buscar los nodos por la propiedad name. Posteriormente se crea la relación entre los nodos a y b una vez sean encontrados.

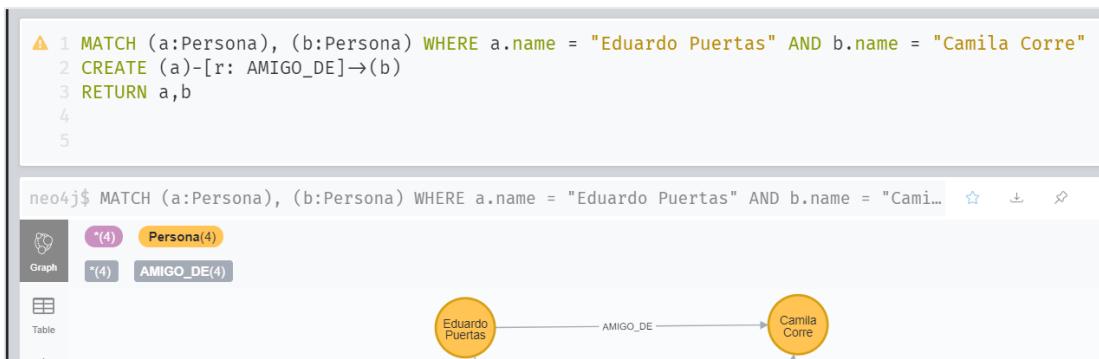


Figura 16. Representación de relación entre nodos y su consulta con CQL.

Otro ejemplo de creación de relación creando previamente y por separado los nodos para relacionar:

- ▶ Primero crea los nodos para relacionar:

```

CREATE (Po:Persona{name:"Po", age:43, nacionality:"Española"}),
(Alan:Persona{name:"Alambrito", age:31, nacionality:"Indio"})

```

- ▶ Luego ejecuta la búsqueda y crea la relación entre los nodos recién creados:

```

MATCH (a:Persona), (b:Persona) WHERE a.name = "Alambrito" AND b.name = "Po"
CREATE (a)-[r:AMIGO_DE {mejorAmigo: "Sí", debeSaldo: 190.50}]->(b)
RETURN a,b

```

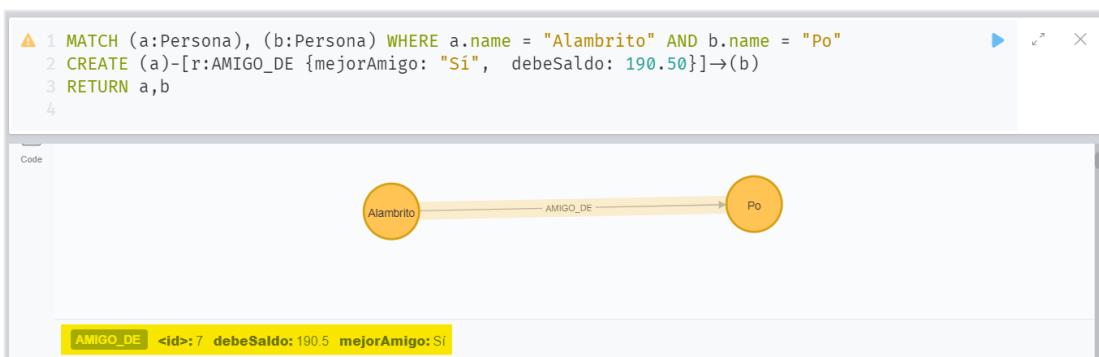


Figura 17. Representación de relación entre nodos, su consulta con CQL y la visualización de las propiedades.

Cláusula Match

- ▶ `MATCH (n) RETURN n`: consulta todos los nodos.
- ▶ `MATCH (n:Persona) RETURN n`: consulta todos los nodos con la etiqueta Persona.
- ▶ `MATCH (Pepe:Persona{name: "Pepe Rueda"})<-[: AMIGO_DE]-(n)`
`RETURN n.name`: retorna la lista de nodos cuya relación con Pepe:Persona es AMIGO_DE.
- ▶ `MATCH (n) detach delete n`: borra todos los nodos existentes.

Otras cláusulas

- ▶ `MATCH (n) RETURN n.name, n.age ORDER BY n.age ASC LIMIT 10`
- ▶ `MATCH (n) RETURN n.name, n.age ORDER BY n.age ASC LIMIT toInt(3 * rand())+ 1`: uso de funciones dentro de las consultas.
- ▶ `MATCH (n) RETURN n.name, n.age ORDER BY n.age ASC SKIP 2`
- ▶ `MATCH (n) WITH n ORDER BY n.name ASC LIMIT 5 RETURN collect(n.name)`: devuelve el resultado de la consulta en forma de colección.
- ▶ `MATCH (Pepe:Persona{name: "Pepe Rueda"}) REMOVE Pepe.nationality RETURN Pepe`: remover una propiedad (age) del nodo.
- ▶ `MATCH (Pepe:Persona{name: "Pepe Rueda"}) REMOVE Pepe:Persona RETURN Pepe`: borrar la etiqueta de un nodo. Puede añadir más de una etiqueta separándolas con dos puntos (:).
- ▶ `MATCH (Persona) WHERE Persona.age = 25 RETURN Persona`: la cláusula WHERE permite acotar la consulta añadiendo condiciones.
- ▶ `MATCH (Persona) WHERE Persona.age > 18 AND Persona.age < 30 RETURN Persona`
- ▶ **Importante:** `MATCH (n) WHERE (n)-[: AMIGO_DE]->({name: "Po"}) RETURN n`: consultar los amigos de Po (Persona).

Llegados a este punto, cuentas con suficientes nociones para revisar otra documentación de Neo4j y, con ello, además de aprender más sobre las cláusulas antes vistas, encontrarás muchas otras que quedan por descubrir y que te darán una base sólida para trabajar con este tipo de bases de datos.

Al final de este tema tienes varios enlaces que te serán útiles para emprender este camino y mejorar tus conocimientos sobre esta forma de almacenar los datos.

Iniciar las bases de datos de ejemplo

Antes de comenzar a escribir tus propias consultas, es recomendable cargar y visualizar algunas de las bases de datos que provee Neo4j como ejemplo. La más conocida es la base de datos de películas, usada en las imágenes anteriores. Para ello, visualice la guía de instalación escribiendo la siguiente instrucción:

```
$ :play movie graph
```

De inmediato se desplegarán las instrucciones para cargar la base de datos. Avanza en los distintos pasos con el botón que se muestra a la derecha del panel visualizado. A lo largo de cada paso, tendrás que ir ejecutando las distintas consultas que se te proponen en pantalla. No te dejes ninguna sin ejecutar y procura seguir el orden propuesto por la interfaz.

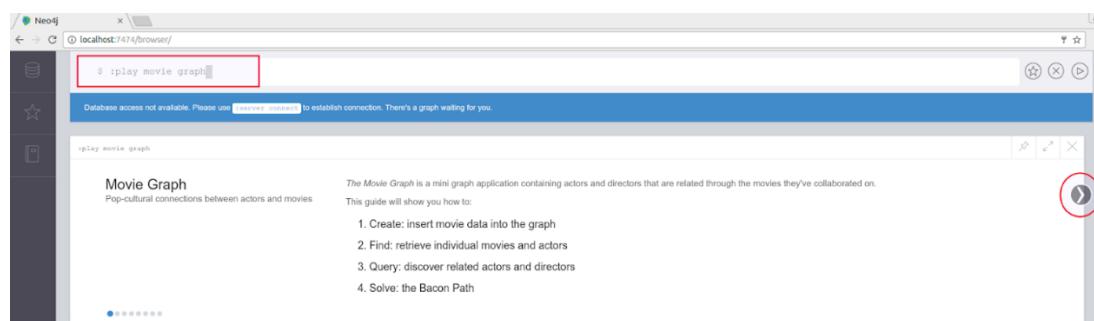


Figura 18. Carga de datos de ejemplo en Neo4j (Movie).

Otro ejemplo de base de datos es el siguiente:

```
$ :play northwind graph
```

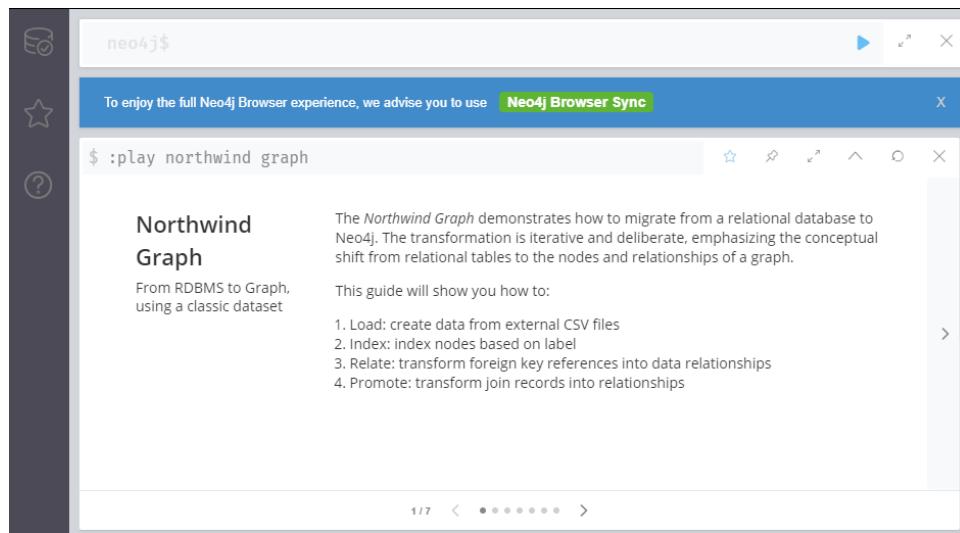


Figura 19. Carga de datos de ejemplo en Neo4j (Northwind).

Recuerda avanzar a través de los distintos pasos e ir ejecutando las instrucciones propuestas en el mismo orden.

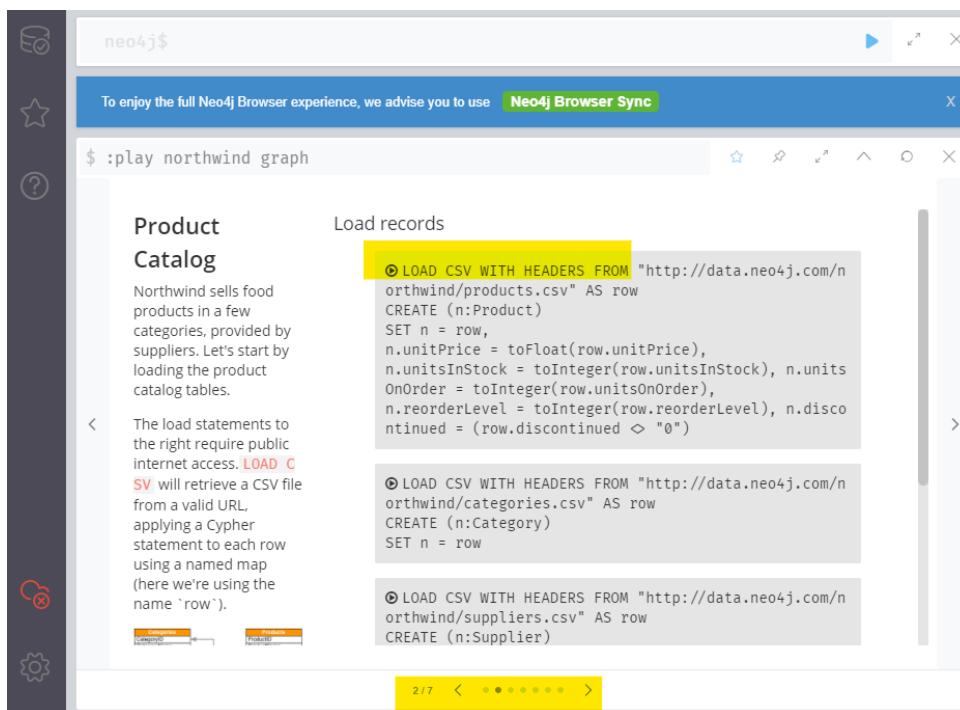


Figura 20. Instrucciones de carga de datos de ejemplo en Neo4j.

8.7. Visualización de grafos

La principal característica de Neo4j es su modo de visualizar los datos y las relaciones que existen entre ellos. El uso de nodos y relaciones permite modelar fácilmente los datos y presentarlos de una forma que tanto los desarrolladores, los analistas y los de negocio puedan entender por igual.

La visualización de grafos potencia esta capacidad de Neo4j permitiendo dibujar el gráfico en varios formatos para que los usuarios puedan interactuar con los datos de una manera más fácil.

Con las herramientas de visualización, un grafo completo o parcial puede cobrar vida y permitir al usuario explorarlo, estableciendo varias reglas o vistas para analizarlo desde diferentes perspectivas.

Existe una variedad de herramientas de código abierto que permiten esta manipulación de los grafos. Cada herramienta proporciona determinadas capacidades, por ello debes elegir la que más se adapte a las necesidades del caso de uso.

Formatos del grafo

Hay una variedad de formatos a la hora de visualizar los datos en Neo4j. Los resultados de una consulta se pueden guardar como JSON, XML, formatos tabulares y formato visual. Este último proporciona un valor adicional para los analistas de datos y los usuarios de negocio, así como para el resto de los usuarios.

En el ejemplo siguiente, de forma visual y casi inmediata, es posible describir los datos que se muestran, porque se observan directamente las relaciones y los nodos que componen el resultado de la consulta.

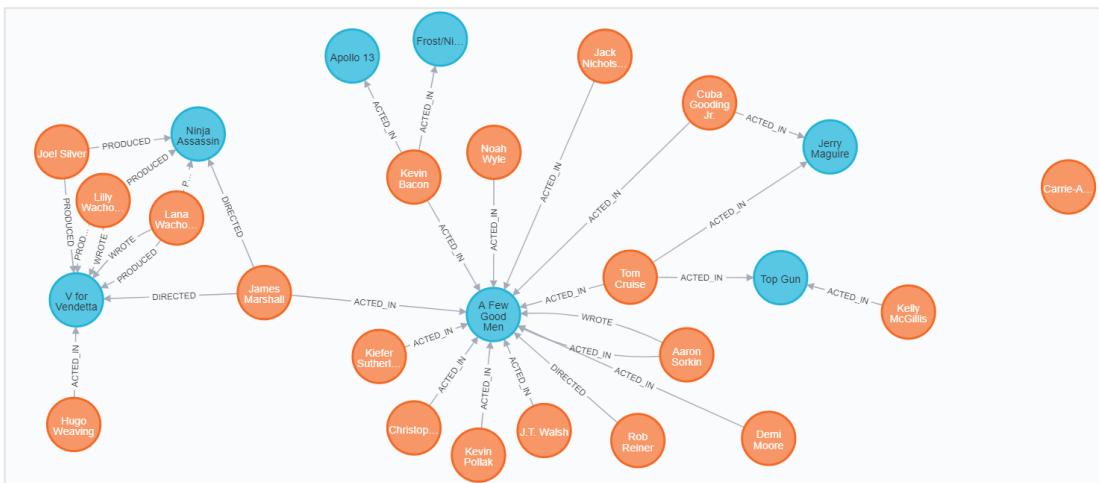


Figura 21. Visualización de grafos en Neo4j.

Neo4j tiene dos herramientas de visualización principales que están construidas y diseñadas para trabajar específicamente con datos en la base de datos de grafos de Neo4j: **Neo4j Browser** y **Neo4j Bloom**.

Neo4j Browser

Está pensada para que los desarrolladores puedan ejecutar sus consultas Cypher y visualizar los resultados de forma inmediata y con la posibilidad de hacerlo desde cualquier ordenador o navegador. Las versiones Enterprise y Community integran esta interfaz de visualización.

También proporciona algunas funciones para diseñar con colores y tamaños según las etiquetas de los nodos y los tipos de relaciones, permitiendo al usuario personalizar sus propios estilos importando un archivo GRASS (hoja de estilo de grafos). Asimismo, facilita la exportación del grafo a formatos PNG, SVG o CSV.

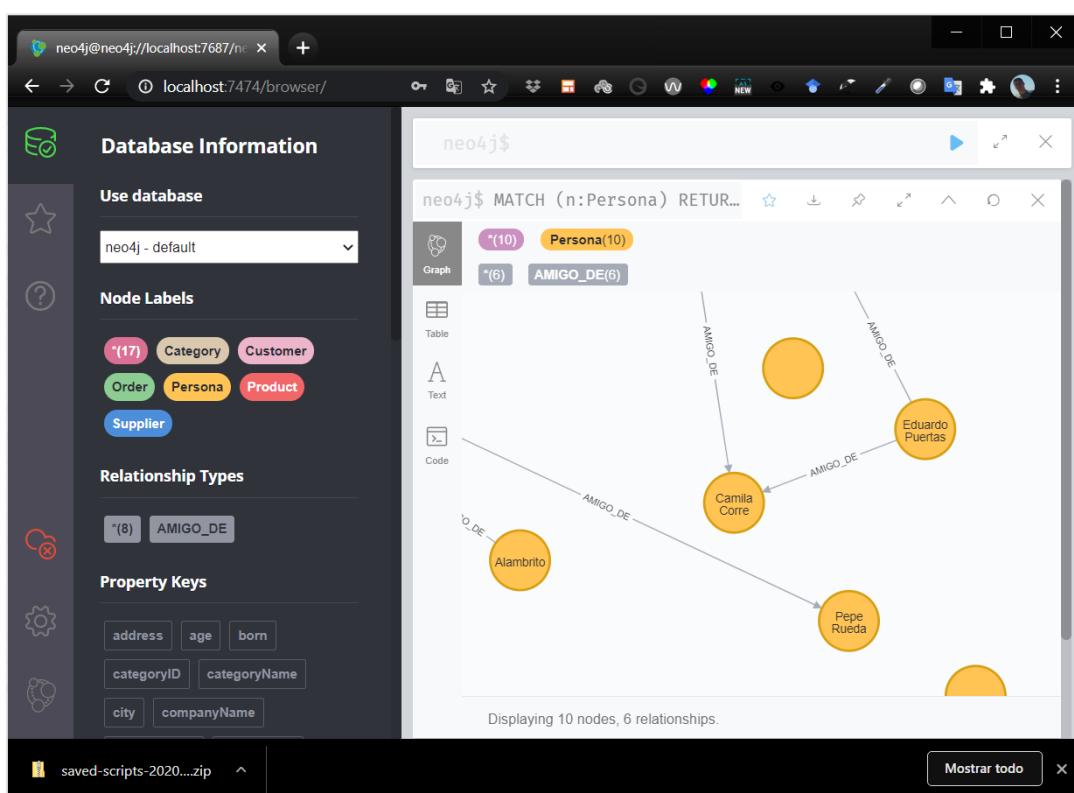


Figura 22. Información de la base de datos en Neo4j desde la web.

Neo4j Bloom

Otro producto externo a Neo4j que permite la visualización de grafos es Bloom. Opera con licencia comercial o de forma gratuita con Neo4j Desktop siempre que sea un único usuario el que lo use. Bloom fue diseñada para que los analistas de negocios y otros no desarrolladores interactúen con los datos almacenados en la base de datos de gráficos sin escribir ningún código.

Los usuarios pueden utilizar el lenguaje natural para consultar la base de datos y explorar patrones, clústeres y recorridos en sus datos gráficos. Permite, a su vez, crear diferentes disecciones del gráfico (perspectivas) que permiten a los usuarios ver diferentes aspectos y porciones de datos del gráfico para su posterior análisis.

Consulta la documentación oficial de Neo4j para encontrar referencias sobre esta herramienta, su enlace [aquí](#).

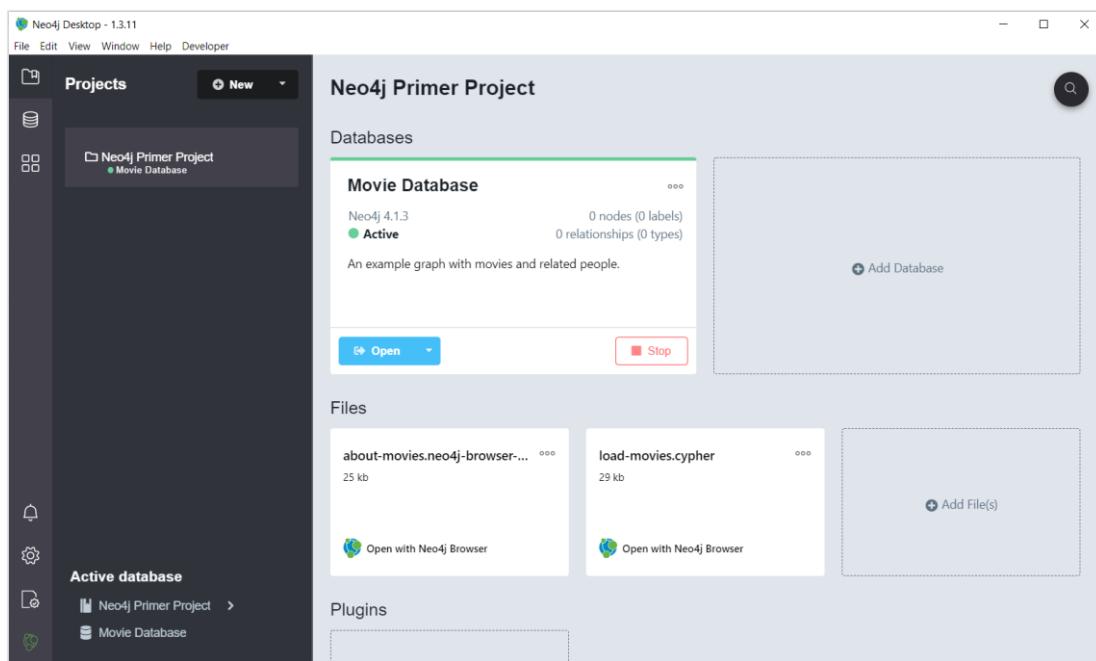


Figura 23. Visualización de la base de datos creada de la web de Neo4j.

Consulta [aquí](#) más herramientas útiles para la visualización de los grafos.

8.8. Caso práctico

El siguiente caso práctico está relacionado con el análisis de datos para la detección de fraude. Para ello, el estudiante irá ejecutando una serie de instrucciones con el fin de mostrarle cómo es posible construir un escenario donde, a través de CQL, se pueden ejecutar consultas que buscan relaciones o patrones en los datos con características puntuales, en este caso: indicios de fraude con tarjetas de crédito.

Importante: la idea principal de este ejemplo fue tomado de la base de casos públicos de Neo4j. El autor de este caso en concreto es Jean Villedieu.

¡Manos a la obra!

- ▶ **Paso 1:** con las siguientes instrucciones crea los nodos Persona necesarios para el análisis.

```
CREATE (Pablo:Persona {id:'1', nombre:'Pablo', sexo:'hombre', edad:'50'})  
CREATE (Jairo:Persona {id:'2', nombre:'Jairo', sexo:'hombre', edad:'48'})  
CREATE (Daniel:Persona {id:'3', nombre:'Daniel', sexo:'hombre',  
edad:'23'})  
CREATE (Marcos:Persona {id:'4', nombre:'Marcos', sexo:'hombre',  
edad:'30'})  
CREATE (John:Persona {id:'5', nombre:'John', sexo:'hombre', edad:'31'})  
CREATE (Zoila:Persona {id:'6', nombre:'Zoila', sexo:'mujer', edad:'52'})  
CREATE (Maria:Persona {id:'7', nombre:'Maria', sexo:'mujer', edad:'23'})  
CREATE (Isabel:Persona {id:'8', nombre:'Isabel', sexo:'mujer', edad:'58'})  
CREATE (Patricia:Persona {id:'9', nombre:'Patricia', sexo:'mujer',  
edad:'51'})  
CREATE (Marina:Persona {id:'10', nombre:'Marina', sexo:'mujer',  
edad:'37'})
```

- **Paso 2:** con las siguientes instrucciones crea los nodos Comercio.

```
CREATE (Amazon:Comercio {id:'11', nombre:'Amazon', calle:'2626 Wilkinson Court', direccion:'San Bernardino, CA 92410'})  
CREATE (Abercrombie:Comercio {id:'12', nombre:'Abercrombie', calle:'4355 Walnut Street', edad:'San Bernardino, CA 92410'})  
CREATE (Walmart:Comercio {id:'13', nombre:'Walmart', calle:'2092 Larry Street', edad:'San Bernardino, CA 92410'})  
CREATE (MacDonalds:Comercio {id:'14', nombre:'MacDonalds', calle:'1870 Caynor Circle', edad:'San Bernardino, CA 92410'})  
CREATE (American_Apparel:Comercio {id:'15', nombre:'American Apparel', calle:'1381 Spruce Drive', edad:'San Bernardino, CA 92410'})  
CREATE (Just_Brew_It:Comercio {id:'16', nombre:'Just Brew It', calle:'826 Anmoore Road', edad:'San Bernardino, CA 92410'})  
CREATE (Justice:Comercio {id:'17', nombre:'Justice', calle:'1925 Spring Street', edad:'San Bernardino, CA 92410'})  
CREATE (Sears:Comercio {id:'18', nombre:'Sears', calle:'4209 Elsie Drive', edad:'San Bernardino, CA 92410'})  
CREATE (Soccer_for_the_City:Comercio {id:'19', nombre:'Soccer for the City', calle:'86 D Street', edad:'San Bernardino, CA 92410'})  
CREATE (Sprint:Comercio {id:'20', nombre:'Sprint', calle:'945 Kinney Street', edad:'San Bernardino, CA 92410'})  
CREATE (Starbucks:Comercio {id:'21', nombre:'Starbucks', calle:'3810 Apple Lane', edad:'San Bernardino, CA 92410'})  
CREATE (Subway:Comercio {id:'22', nombre:'Subway', calle:'3778 Tenmile Road', edad:'San Bernardino, CA 92410'})  
CREATE (Apple_Store:Comercio {id:'23', nombre:'Apple Store', calle:'349 Bel Meadow Drive', edad:'Kansas City, MO 64105'})  
CREATE (Urban_Outfitters:Comercio {id:'24', nombre:'Urban Outfitters', calle:'99 Strother Street', edad:'Kansas City, MO 64105'})  
CREATE (RadioShack:Comercio {id:'25', nombre:'RadioShack', calle:'3306 Douglas Dairy Road', edad:'Kansas City, MO 64105'})  
CREATE (Macys:Comercio {id:'26', nombre:'Macys', calle:'2912 Nutter Street', edad:'Kansas City, MO 64105'})
```

- **Paso 3:** crea las siguientes relaciones entre personas y comercios.

```

CREATE  (Pablo)-[:COMPRO_EN  {pago:'986',    fecha:'4/17/2014',    estado:
'Indiscutible'}]->(Just_Brew_It)
CREATE  (Pablo)-[:COMPRO_EN  {pago:'239',    fecha:'5/15/2014',    estado:
'Indiscutible'}]->(Starbucks)
CREATE  (Pablo)-[:COMPRO_EN  {pago:'475',    fecha:'3/28/2014',    estado:
'Indiscutible'}]->(Sears)
CREATE  (Pablo)-[:COMPRO_EN  {pago:'654',    fecha:'3/20/2014',    estado:
'Indiscutible'}]->(Wallmart)
CREATE  (Jairo)-[:COMPRO_EN  {pago:'196',    fecha:'7/24/2014',    estado:
'Indiscutible'}]->(Soccer_for_the_City)
CREATE  (Jairo)-[:COMPRO_EN  {pago:'502',    fecha:'4/9/2014',    estado:
'Indiscutible'}]->(Abercrombie)
CREATE  (Jairo)-[:COMPRO_EN  {pago:'848',    fecha:'5/29/2014',    estado:
'Indiscutible'}]->(Wallmart)
CREATE  (Jairo)-[:COMPRO_EN  {pago:'802',    fecha:'3/11/2014',    estado:
'Indiscutible'}]->(Amazon)
CREATE  (Jairo)-[:COMPRO_EN  {pago:'203',    fecha:'3/27/2014',    estado:
'Indiscutible'}]->(Subway)
CREATE  (Daniel)-[:COMPRO_EN  {pago:'35',     fecha:'1/23/2014',    estado:
'Indiscutible'}]->(MacDonalds)
CREATE  (Daniel)-[:COMPRO_EN  {pago:'605',    fecha:'1/27/2014',    estado:
'Indiscutible'}]->(MacDonalds)
CREATE  (Daniel)-[:COMPRO_EN  {pago:'62',     fecha:'9/17/2014',    estado:
'Indiscutible'}]->(Soccer_for_the_City)
CREATE  (Daniel)-[:COMPRO_EN  {pago:'141',    fecha:'11/14/2014',    estado:
'Indiscutible'}]->(Amazon)
CREATE  (Marcos)-[:COMPRO_EN  {pago:'134',    fecha:'4/14/2014',    estado:
'Indiscutible'}]->(Amazon)
CREATE  (Marcos)-[:COMPRO_EN  {pago:'336',    fecha:'4/3/2014',    estado:
'Indiscutible'}]->(American_Apparel)
CREATE  (Marcos)-[:COMPRO_EN  {pago:'964',    fecha:'3/22/2014',    estado:
'Indiscutible'}]->(Wallmart)
CREATE  (Marcos)-[:COMPRO_EN  {pago:'430',    fecha:'8/10/2014',    estado:
'Indiscutible'}]->(Sears)
CREATE  (Marcos)-[:COMPRO_EN  {pago:'11',     fecha:'9/4/2014',    estado:
'Indiscutible'}]->(Soccer_for_the_City)

```

```

CREATE (John)-[:COMPRO_EN {pago:'545', fecha:'10/6/2014', estado:'Indiscutible'}]->(Soccer_for_the_City)
CREATE (John)-[:COMPRO_EN {pago:'457', fecha:'10/15/2014', estado:'Indiscutible'}]->(Sprint)
CREATE (John)-[:COMPRO_EN {pago:'468', fecha:'7/29/2014', estado:'Indiscutible'}]->(Justice)
CREATE (John)-[:COMPRO_EN {pago:'768', fecha:'11/28/2014', estado:'Indiscutible'}]->(American_Apparel)
CREATE (John)-[:COMPRO_EN {pago:'921', fecha:'3/12/2014', estado:'Indiscutible'}]->(Just_Brew_It)
CREATE (Zoila)-[:COMPRO_EN {pago:'740', fecha:'12/15/2014', estado:'Indiscutible'}]->(MacDonalds)
CREATE (Zoila)-[:COMPRO_EN {pago:'510', fecha:'11/27/2014', estado:'Indiscutible'}]->(Abercrombie)
CREATE (Zoila)-[:COMPRO_EN {pago:'414', fecha:'1/20/2014', estado:'Indiscutible'}]->(Just_Brew_It)
CREATE (Zoila)-[:COMPRO_EN {pago:'721', fecha:'7/17/2014', estado:'Indiscutible'}]->(Amazon)
CREATE (Zoila)-[:COMPRO_EN {pago:'353', fecha:'10/25/2014', estado:'Indiscutible'}]->(Subway)
CREATE (Maria)-[:COMPRO_EN {pago:'681', fecha:'12/28/2014', estado:'Indiscutible'}]->(Sears)
CREATE (Maria)-[:COMPRO_EN {pago:'87', fecha:'2/19/2014', estado:'Indiscutible'}]->(Wallmart)
CREATE (Maria)-[:COMPRO_EN {pago:'533', fecha:'8/6/2014', estado:'Indiscutible'}]->(American_Apparel)
CREATE (Maria)-[:COMPRO_EN {pago:'723', fecha:'1/8/2014', estado:'Indiscutible'}]->(American_Apparel)
CREATE (Maria)-[:COMPRO_EN {pago:'627', fecha:'5/20/2014', estado:'Indiscutible'}]->(Just_Brew_It)
CREATE (Isabel)-[:COMPRO_EN {pago:'74', fecha:'9/4/2014', estado:'Indiscutible'}]->(Soccer_for_the_City)
CREATE (Isabel)-[:COMPRO_EN {pago:'231', fecha:'7/12/2014', estado:'Indiscutible'}]->(Wallmart)
CREATE (Isabel)-[:COMPRO_EN {pago:'924', fecha:'10/4/2014', estado:'Indiscutible'}]->(Soccer_for_the_City)
CREATE (Isabel)-[:COMPRO_EN {pago:'742', fecha:'8/12/2014', estado:'Indiscutible'}]->(Just_Brew_It)
CREATE (Patricia)-[:COMPRO_EN {pago:'276', fecha:'12/24/2014', estado:'Indiscutible'}]->(Soccer_for_the_City)

```

```

CREATE      (Patricia)-[:COMPRO_EN] {pago:'66', fecha:'4/16/2014',
estado:'Indiscutible'}]->(Starbucks)
CREATE      (Patricia)-[:COMPRO_EN] {pago:'467', fecha:'12/23/2014',
estado:'Indiscutible'}]->(MacDonalds)
CREATE      (Patricia)-[:COMPRO_EN] {pago:'830', fecha:'3/13/2014',
estado:'Indiscutible'}]->(Sears)
CREATE      (Patricia)-[:COMPRO_EN] {pago:'240', fecha:'7/9/2014',
estado:'Indiscutible'}]->(Amazon)
CREATE      (Patricia)-[:COMPRO_EN] {pago:'164', fecha:'12/26/2014',
estado:'Indiscutible'}]->(Soccer_for_the_City)
CREATE      (Marina)-[:COMPRO_EN] {pago:'630', fecha:'10/6/2014',
estado:'Indiscutible'}]->(MacDonalds)
CREATE      (Marina)-[:COMPRO_EN] {pago:'19', fecha:'7/29/2014',
estado:'Indiscutible'}]->(Abercrombie)
CREATE      (Marina)-[:COMPRO_EN] {pago:'352', fecha:'12/16/2014',
estado:'Indiscutible'}]->(Subway)
CREATE      (Marina)-[:COMPRO_EN] {pago:'147', fecha:'8/3/2014',
estado:'Indiscutible'}]->(Amazon)
CREATE      (Marina)-[:COMPRO_EN] {pago:'91', fecha:'6/29/2014',
estado:'Indiscutible'}]->(Wallmart)
CREATE      (Pablo)-[:COMPRO_EN] {pago:'1021', fecha:'7/18/2014',
estado:'Cuestionable'}]->(Apple_Store)
CREATE      (Pablo)-[:COMPRO_EN] {pago:'1732', fecha:'5/10/2014',
estado:'Cuestionable'}]->(Urban_Outfitters)
CREATE      (Pablo)-[:COMPRO_EN] {pago:'1415', fecha:'4/1/2014',
estado:'Cuestionable'}]->(RadioShack)
CREATE      (Pablo)-[:COMPRO_EN] {pago:'1849', fecha:'12/20/2014',
estado:'Cuestionable'}]->(Macys)
CREATE      (Marcos)-[:COMPRO_EN] {pago:'1914', fecha:'7/18/2014',
estado:'Cuestionable'}]->(Apple_Store)
CREATE      (Marcos)-[:COMPRO_EN] {pago:'1424', fecha:'5/10/2014',
estado:'Cuestionable'}]->(Urban_Outfitters)
CREATE      (Marcos)-[:COMPRO_EN] {pago:'1721', fecha:'4/1/2014',
estado:'Cuestionable'}]->(RadioShack)
CREATE      (Marcos)-[:COMPRO_EN] {pago:'1003', fecha:'12/20/2014',
estado:'Cuestionable'}]->(Macys)
CREATE      (Isabel)-[:COMPRO_EN] {pago:'1149', fecha:'7/18/2014',
estado:'Cuestionable'}]->(Apple_Store)
CREATE      (Isabel)-[:COMPRO_EN] {pago:'1152', fecha:'8/10/2014',
estado:'Cuestionable'}]->(Urban_Outfitters)

```

```
CREATE      (Isabel)-[:COMPRO_EN]-(r:{pago:'1884', fecha:'8/1/2014', estado:'Cuestionable'})->(RadioShack)
CREATE      (Isabel)-[:COMPRO_EN]-(r:{pago:'1790', fecha:'12/20/2014', estado:'Cuestionable'})->(Macys)
CREATE      (Marina)-[:COMPRO_EN]-(r:{pago:'1925', fecha:'7/18/2014', estado:'Cuestionable'})->(Apple_Store)
CREATE      (Marina)-[:COMPRO_EN]-(r:{pago:'1374', fecha:'7/10/2014', estado:'Cuestionable'})->(Urban_Outfitters)
CREATE      (Marina)-[:COMPRO_EN]-(r:{pago:'1368', fecha:'7/1/2014', estado:'Cuestionable'})->(RadioShack)
CREATE      (Marina)-[:COMPRO_EN]-(r:{pago:'1816', fecha:'12/20/2014', estado:'Cuestionable'})->(Macys)
```

- **Paso 4:** ejecuta la siguiente instrucción. ¿Funciona? ¿Qué arroja la consulta?

```
MATCH (victima:Persona)-[r:COMPRO_EN]->(comercio)
WHERE r.estado = "Cuestionable"
RETURN victima.nombre AS `Cliente`, comercio.nombre AS `Tienda`, r.pago AS Pago, r.fecha AS `Fecha Transaccion`
ORDER BY `Fecha Transaccion` DESC
```

- **Paso 5:** ejecuta la siguiente instrucción. ¿Funciona? ¿Qué arroja esta otra consulta?

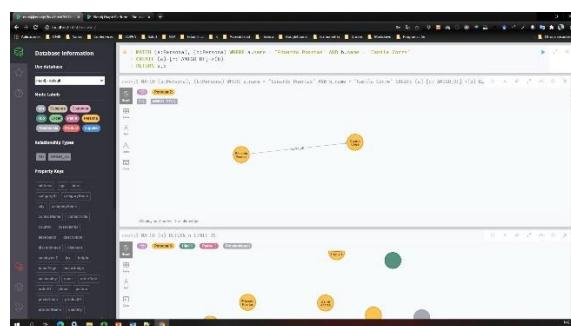
```
MATCH (victima:Persona)-[r:COMPRO_EN]->(comercio)
WHERE r.estado = "Cuestionable"
MATCH (victima)-[t:COMPRO_EN]->(otrocomercio)
WHERE t.estado = "Indiscutible" AND t.fecha < r.fecha
WITH victima, otrocomercio, t ORDER BY t.fecha DESC
RETURN victima.nombre AS `Cliente`, otrocomercio.nombre AS `Tienda`, t.pago AS Pago, t.fecha AS `Fecha Transaccion`
ORDER BY `Fecha Transaccion` DESC
```

- **Paso 6:** ejecuta la siguiente instrucción. ¿Funciona? ¿Qué arroja esta otra consulta?

```
MATCH (victima:Persona)-[r:COMPRO_EN]->(comercio)
WHERE r.estatus = "Cuestionable"
MATCH (victima)-[t:COMPRO_EN]->(otrocomercio)
WHERE t.estado = "Indiscutible" AND t.fecha < r.fecha
WITH victima, otrocomercio, t ORDER BY t.fecha DESC
RETURN DISTINCT otrocomercio.nombre AS `Tienda Sospechosa`, count(DISTINCT t) AS Count, collect(DISTINCT victima.nombre) AS Victimas
ORDER BY Count DESC
```

- Comparte tu avance con el resto de la clase, para ello publica en el Foro de la asignatura (en un único mensaje) tu interpretación de cada una de las consultas.

Después de leer todo el tema, accede a la lección titulada «Uso de Neo4j». En ella se hace un repaso sobre los aspectos más relevantes de la base de datos de grafos y sobre Neo4j. Mediante un caso práctico se describen los nodos, las relaciones y las propiedades.



Vídeo 1. Uso de Neo4j.

Accede al vídeo a través del aula virtual

8.9. Referencias bibliográficas

Fernandes, D., y Bernardino, J. (2018). Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4j, and OrientDB. En J. Bernardino y C. Quix (eds.), *Proceedings of the 7th International Conference on Data Science, Technology and Applications (DATA 2018)* (pp. 373-380). SCITEPRESS.

Guia, J., Gonçalves, V., y Bernardino, J. (2017). Graph databases: Neo4j analysis. En S. Hammoudi, M. Smialek, O. Camp y J. Filipe (eds.), *Proceedings of the 19th International Conference on Enterprise Information Systems (Volume 3)* (pp. 351-356). ICEIS.

Hölsch, J., Schmidt, T., y Grossniklaus, M. (2017). On the performance of analytical and pattern matching graph queries in Neo4j and a relational database. En *EDBT/ICDT 2017 Joint Conference: 6th International Workshop on Querying Graph Structured Data (GraphQ)*.

http://ceur-ws.org/Vol-1810/GraphQ_paper_01.pdf

Lal, M. (2015). *Neo4j graph data modeling*. Packt Publishing.

Webber, J., y Robinson, I. (2018). *A programmatic introduction to Neo4j*. Addison-Wesley Professional.

A fondo

Documentación oficial de Neo4j: características generales de la base de datos

Neo4j. (2021). The Neo4j Getting Started Guide v4.2. *Neo4j docs* [Página web].

<https://neo4j.com/docs/getting-started/current/>

Neo4j es una base de datos que cuenta con una documentación bastante completa y detallada. En su página web la documentación de la herramienta se enfoca en tres aspectos claves: introducción a Neo4j, conceptos de grafos y el lenguaje de consultas CQL.

Tutorial de Neo4j. Introducción: *graph database*

Tutorialspoint. (2021). Learn Neo4j: graph database [Página web].

<https://www.tutorialspoint.com/neo4j/index.htm>

El equipo de Tutorialspoint tiene en su sitio web un tutorial bastante completo de Neo4j, el cual cuenta con muchos ejemplos prácticos que permiten conocer la herramienta de forma amena y sencilla.



Introduction to Neo4j: a hands-on crash course

Neo4j. (26 de noviembre de 2020). *Intro to Neo4j for Developers with Lju* [Archivo de vídeo]. <https://www.youtube.com/watch?v=n2wgFTTZGps>



En el siguiente vídeo es posible ver una introducción a Neo4j y al uso de grafos para representar la información. El vídeo lo ha elaborado parte del equipo de desarrolladores de Neo4j.

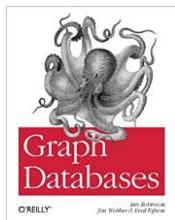
Casos de uso basados en Neo4j: puesta en práctica de la teoría de grafos en escenarios de negocio

Neo4j. (2021). Graph Database Use Cases [Página web]. <https://neo4j.com/use-cases/>

La mejor manera de comprender el uso que se le puede dar a las bases de datos de grafos es mediante ejemplos de casos que muestren el potencial de este tipo de herramientas. La web oficial de Neo4j dispone de varios casos de uso que pueden ser analizados para aprender el dominio de los grafos. Consulta dichos casos y explora los datos y consultas que utilizan para sacar partido a la información representada con nodos, relaciones y propiedades.

Graph Databases: libro oficial recomendado por Neo4j para el estudio de grafos

Neo4j. (2021). Neo4j Download Center [Página web]. <https://neo4j.com/download-center/#community>



Uno de los mejores libros para aprender el concepto de grafos y la utilidad que tienen este tipo de representación de los datos dentro de distintos casos de uso en las organizaciones. Es posible obtener el libro al descargar Neo4j Community.

Bibliografía

Vukotic, A., Watt, N., Abedrabbo, T., Fox, D., & Partner, J. (2015). *Neo4j in action* (Vol. 22). Shelter Island: Manning.

Kokash, N., Romanello, M., Suyver, E., & Colavizza, G. (2022). From Books to Knowledge Graphs. *arXiv preprint arXiv:2204.10766*.

Frisendal, T. (2018). Using GraphQL with an Existing Graph Database. En *Visual Design of GraphQL Data* (pp. 81-91). Apress, Berkeley, CA.

Needham, M., & Hodler, A. E. (2019). *Graph algorithms: practical examples in Apache Spark and Neo4j*. O'Reilly Media.

Jyothi, D. N. (2020). Book Recommendation System using Neo4j Graph Database. The *International Journal of Analytical and Experimental Modal Analysis*, 12(6), 498-504.

1. ¿Cuáles son los principales componentes de un grafo?

- A. El esquema, los nodos, las relaciones y las propiedades.
- B. Nodos, relaciones y propiedades.
- C. Nodos y relaciones, las propiedades están implícitas en cada uno.
- D. Ninguna de las anteriores es correcta.

Un grafo se representa principalmente por nodos, relaciones y propiedades. Otros elementos, como los esquemas, son propios del concepto de base de datos.

2. ¿Qué afirmación define las características principales del modelo de grafos?

- A. El modelo representa datos en nodos, relaciones y propiedades.
- B. Las propiedades son pares clave-valor.
- C. Las relaciones conectan los nodos.
- D. Todas las anteriores son correctas.

Todas las afirmaciones definen las características de Neo4j vistas en este tema.

3. ¿Qué es CQL?

- A. El lenguaje de consulta para Neo4j Graph Database.
- B. Un lenguaje para insertar y borrar nodos y relaciones.
- C. Un lenguaje enfocado a encontrar solo nodos dentro de un grafo.
- D. Todas las afirmaciones anteriores son correctas.

CQL o Cypher Query Language es el lenguaje que utiliza Neo4j para la manipulación de grafos.

4. ¿Qué hace la siguiente instrucción: MATCH (n) DETACH DELETE n?

- A. Limpia la base de datos.
- B. Borra todos los nodos y relaciones de la base de datos.
- C. Borra solo los nodos existentes.
- D. Borra los nodos y sus propiedades.

Cuidado, esta instrucción borra tanto nodos como relaciones de la base de datos. De igual forma, incluye las propiedades de ambos elementos.

5. La cláusula que borra etiquetas y propiedades es:

- A. DELETE.
- B. UNWIND.
- C. REMOVE.
- D. DROP.

REMOVE permite eliminar tanto las etiquetas como las propiedades de nodos y relaciones. DELETE borra nodos o relaciones y DROP no existe en Neo4j.

6. ¿Qué significa n en la siguiente instrucción: MATCH (n) RETURN n.name, n.runs ORDER BY n.runs?

- A. Cualquier nodo y relación.
- B. Cualquier nodo.
- C. Cualquier relación con las propiedades *name* y *runs*.
- D. Los nodos cuya propiedad *name* tiene un valor.

n representa cualquier nodo relacionado con otros nodos.

7. ¿La instruction MATCH (n) RETURN n.name order by n.name qué retorna?

- A. Un grafo con los nodos *n*.
- B. Un listado.
- C. Un listado y un grafo.
- D. Una lista ordenada de las propiedades nombre de todos los nodos.

Al igual que en SQL, *order by* ordena un resultado, en este caso el retorno en la lista de nombres ordenada ascendentemente por defecto.

8. ¿La instrucción CREATE (Pepe)-[r:CANTA_EN{name="Padro"}]-(Concierto) genera algún tipo de error al ser ejecutada?

- A. No, crea los nodos Pepe y Concierto y su relación CANTA_EN.
- B. Sí, la propiedad de la relación es incorrecta.
- C. Sí, no se indica una dirección o sentido de la relación.
- D. Todas las respuestas anteriores son correctas.

Siempre es necesario indicar el sentido de la relación. Si este no se indica, Neo4j genera un error preguntando por esta definición.

9. La instrucción CREATE (Juan)-[r:HIJO_DE]->(Alma) crea:

- A. Los nodos Juan y Alma y una relación entre ellos llamada HIJO_DE.
- B. Una relación entre los nodos Juan y Alma llamada HIJO_DE.
- C. Una relación saliente de Juan hacia Ana llamada HIJO_DE.
- D. Ninguna de las anteriores es correcta

Juan y Alma son nodos, aunque existan se crearán dichos nodos y ambos se le asignará la relación HIJO_DE.

10. Indica qué hace la siguiente instrucción: MATCH (a:Jugador), (b:Juego) WHERE a.name = "Rabino" AND b.name = "Casino" CREATE (a)-[r:TRABAJA_EN {partida:1, victorias:5}]->(b) RETURN a,b

- A. Crea los nodos a y b y asigna la relación TRABAJA_EN.
- B. Crea la relación TRABAJA_EN sobre los nodos a y b.
- C. Busca los nodos a y b que cumplan la condición del WHERE y luego asigna la relación TRABAJA_EN entre ellos.
- D. Todas las respuestas anteriores son correctas.

Al ejecutarse juntas MATCH y CREATE, los nodos que cumplen la condición WHERE serán quienes reciban la relación que se indica en el CREATE.

Bases de Datos para el Big Data

Redis

Índice

Esquema	3
Ideas clave	4
9.1. Introducción y objetivos	4
9.2. Conceptos generales	5
9.3. Modelo de datos, estructura y módulos	8
9.4. Arquitectura y topología	18
9.5. Despliegue en Docker	20
9.6. <i>Drivers</i>	25
9.7. Referencias bibliográficas	27
A fondo	28
Test	30

Esquema

Redis	
Definición	Principales conceptos
Redis-cli	Clúster, nodo, tipos de datos, estructuras y módulos
<p>Base de datos multimodelo:</p> <ul style="list-style-type: none"> - Redis proporciona una funcionalidad completa de varios modelos a través de sus módulos. El uso de Redis como una base de datos de múltiples modelos permite una mayor flexibilidad para los desarrolladores de aplicaciones dentro de una organización. 	
<p>Almacenamiento en memoria:</p> <ul style="list-style-type: none"> - Redis mantiene los datos en la memoria para un acceso rápido y persiste los datos en el almacenamiento, así como la replicación de los contenidos en la memoria para escenarios de producción de alta disponibilidad. 	<p>Estructuras de datos:</p> <ul style="list-style-type: none"> - <i>Strings.</i> - <i>Lists.</i> - <i>Sets.</i> - <i>Sorted sets.</i> - <i>Hashes.</i> - <i>Bit arrays.</i> - <i>Streams.</i> - <i>HyperLogLogs.</i>
<p>Base de datos NoSQL de tipo clave-valor en memoria</p>	<p>Casos de uso/estructura:</p> <ul style="list-style-type: none"> - Cada estructura de datos tiene un caso de uso o escenario diferente para el que se adapta mejor. - Además de las estructuras de datos, Redis también admite el patrón Publicar/Suscribir (Pub/Sub) y patrones adicionales que hacen que sea adecuada para aplicaciones modernas con uso intensivo de datos.
<p>Cómo aplica el teorema CAP</p>	<p>Los principales elementos del modelo de datos Redis son:</p> <ul style="list-style-type: none"> - Tipos de datos. - Estructuras. - Módulos.
<p>redislabs HOME OF REDIS</p>	
	<p>Módulos</p> <ul style="list-style-type: none"> - Redis Graph. - RedisSearch. - Redis TimeSeries. - RedisJSON. <p>Drivers</p> <ul style="list-style-type: none"> - Python, Java, PHP, NodeJS, C, C#. <p>Instalación entorno de test con Docker</p>

9.1. Introducción y objetivos

Este tema es una guía de introducción para entender el funcionamiento de Redis y conocer por qué es tan popular como base de datos NoSQL. Para estudiar este tema, debes leer las Ideas clave y, si deseas información adicional sobre un concepto específico, puedes consultar las lecturas citadas en «Referencias bibliográficas».

Para darle un enfoque práctico a este tema, se recomienda llevar a cabo la instalación y configuración de la herramienta en un entorno local. Para ello, sigue las instrucciones del apartado 9.5 y luego continúa con los otros apartados propuestos. Conforme vayas viendo los distintos comandos podrás ponerlos en práctica en tu propia instalación de Redis.

Los objetivos que se pretenden conseguir con este tema son los siguientes:

- ▶ Conocer los conceptos más generales que describen las características de la base de datos Redis y su funcionamiento.
- ▶ Entender la arquitectura y cómo esta provee de sencillez al motor de base de datos para integrarse con otras herramientas.
- ▶ Repasar las principales estructuras de datos, que son clave en el buen desempeño de Redis como base de datos NoSQL en memoria.
- ▶ Comprender los distintos escenarios donde Redis se adapta para dar solución a aquellos casos donde el acceso al dato debe ser inmediato.
- ▶ Aprender a instalar Redis y a configurar de forma básica un servidor local de pruebas con una base de datos operativa sobre la que trabajar.

9.2. Conceptos generales

Redis surgió como un simple diccionario remoto (**R**Emote **D**ictionary **S**erver). Actualmente se define como una base de datos multimodelo que permite búsquedas, mensajería, transmisión, gráficos y otras funcionalidades más que superan su definición de almacén de datos. Dentro del gran número de bases de datos NoSQL se encuentra Redis como una de las más populares y más utilizada dentro de la tipología de bases de datos clave-valor. Este uso masivo de Redis se debe principalmente a su velocidad, la cual es comprensible, porque trabaja con la información en memoria.



Figura 1. Logotipo comercial de Redis. Fuente: <https://redislabs.com/>

Además de la sencillez en su uso y su flexibilidad, también permite manipular tipos de datos con gran soltura pese a su tipología (clave-valor). Redis permite asociar valores de tipo *string* a una clave y facilita el uso de tipos de datos avanzados, los cuales encajan bastante bien en muchos casos de uso.

Desde que se piensa en el uso de base de datos NoSQL, se da un valor importante a la especialización en la gestión de datos. Redis precisamente es una de esas soluciones que suele encajar como motor de almacenamiento, pero que busca integrarse con otras bases de datos NoSQL y relacionales para sacar el mejor provecho al tratamiento de la información. Esta es la razón por la que Redis se integra en muchos entornos e incluso en grandes soluciones del *cloud*.

Características de Redis

Las estructuras de datos son la clave principal de Redis. Esta base de datos soporta varias estructuras de datos, las cuales se almacenan de tal forma que el concepto clave-valor va un poco más allá. Las estructuras de datos que soporta Redis son:

- ▶ *String.*
- ▶ *Lists.*
- ▶ *Sets.*
- ▶ *Sorted sets.*
- ▶ *Hashes.*
- ▶ *Bit arrays.*
- ▶ *Streams.*
- ▶ *HyperLogLogs.*

Estas estructuras de datos no se usan de forma genérica, sino pensando en un caso de uso concreto, y ahí radica la estrategia de Redis a la hora de proponerse como una base de datos NoSQL. Otras características de Redis son las siguientes:

- ▶ Soporta el patrón publicar/suscribir (pub/sub).
- ▶ Se ejecuta del lado del servidor, principalmente en los sistemas operativos basados en Unix como Linux y Mac, y también como un contenedor Docker en Microsoft Windows.
- ▶ La instalación de Redis es sencilla y su disponibilidad es casi inmediata una vez es público.
- ▶ El servidor espera las conexiones de los clientes, bien sea de forma programática o a través de la interfaz de línea de comandos (CLI).
- ▶ Como toda base de datos NoSQL, no es necesario definir un modelo de datos para poder trabajar con Redis.
- ▶ Lo anterior ocurre también con la creación de tablas; estas no son necesarias para utilizar y sacar provecho de la base de datos.

- ▶ El comando SET se utiliza para crear datos dentro de la base de datos actual, sin la necesidad de definir una estructura previa.

La instalación de Redis está muy documentada, consulta esta [guía](#).

Flexibilidad de Redis y uso de memoria

Es importante destacar este aspecto de la base de datos. Cuando hay costumbre de diseñar, definir y crear un modelo de datos que soporte nuestro trabajo en la base de datos, Redis puede ser confuso al principio precisamente porque no requiere de esos pasos previos. Sin embargo, justamente esta flexibilidad es la que le brinda la posibilidad de adaptarse a diferentes escenarios para la explotación del dato.

Redis almacena los datos en la memoria de acceso aleatorio (RAM) del servidor donde esté instalado. Conforme los datos crecen, el uso de memoria también se incrementa, igual que ocurre con Spark. En la versión Enterprise, Redis puede usar la memoria *flash* disponible para que la base de datos almacene datos (como una unidad de estado sólido dedicada).

En este escenario, las claves se alojan en la RAM y solo determinados valores se guardan en la memoria *flash*. Si lo observamos de forma lógica, los valores más próximos a ser usados estarán en la RAM, mientras que los menos próximos, en *flash*. LRU es el algoritmo que utiliza Redis para decidir qué valores alojar en una memoria o en otra.

La escritura de la base de datos en disco se lleva a cabo a intervalos variables totalmente configurables. Se tiene en cuenta la cantidad de datos que cambian durante dicho intervalo. En caso de fallo o caída del servidor, la escritura en disco garantiza la durabilidad de la información. En un entorno productivo es común encontrar Redis instalado sobre clústeres con alta disponibilidad.

ACID en Redis

Redis emplea diferentes métodos para cumplir con las propiedades **ACID (Atomicity, Consistency, Isolation y Durability)**. Estos métodos son:

- ▶ **Atomicidad:** proporciona comandos que potencian las transacciones, tales como WATCH, MULTI y EXEC. Estos comandos aseguran que las operaciones en la base de datos sean indivisibles e irreductibles.
- ▶ **Coherencia:** permite realizar solo escrituras autorizadas, mediante la validación o autenticación que proporciona Redis.
- ▶ **Aislamiento:** cada comando o transacción que utiliza MULTI o EXEC queda aislado, porque Redis es de un solo subproceso.
- ▶ **Durabilidad:** es posible confirmar que se ha llevado a cabo una operación de escritura en el disco gracias a la configuración de respuesta a una escritura del cliente.

9.3. Modelo de datos, estructura y módulos

Los modelos de datos representan la forma en que se almacenan estos en una base de datos. Cualquier aplicación que utilice dicho modelo estará sujeta a él para explotar de la menor manera la forma de persistir los datos. Con el auge de las tecnologías NoSQL como Redis, el modelo de datos puede ser un reflejo de la aplicación en sí.

En este sentido, Redis es una base de datos que soporta múltiples modelos de datos para que estos se representen de múltiples formas, permitiendo así representar varios casos de uso simultáneamente.

Pese a los múltiples beneficios de este enfoque, no todo es color rosa a la hora de asumir este enfoque de Redis. Una base de datos de cualquier tipo obliga a pensar y decidir sobre la forma de representar los datos dentro del almacén de datos. La forma que se decida es la que luego definirá cómo se insertarán los datos y cómo se leerán.

Redis almacena los datos mediante claves, las cuales pueden ser de cualquier tipo porque realmente son elementos binarios. Para comprender este concepto, una imagen puede ser usada como clave dentro de la base de datos. Aunque existe esta posibilidad, por lo general las claves son cadenas simples.

Para trabajar con los distintos tipos de datos, Redis dispone de una serie de comandos que permiten manipular dichos datos. **SET** y **GET** son dos de los comandos más usados para tal fin. **SET** crea o cambia un valor que corresponde a una clave determinada. **GET** recupera el valor asociado a una clave determinada.

SET sobrescribe los valores si se llama dos veces para la misma clave. El valor último será el que se almacene con dicha instrucción. Los valores que corresponden a una clave determinada se pueden formatear de muchas formas para crear un modelo de datos específico para las necesidades de la organización.

String y bitmaps

En Redis el tipo de valor más simple es un *string*. Los *string* se pueden agregar como valor a la base de datos mediante el comando **SET**. La sintaxis de **SET** requiere una clave y un valor para crear la entrada.

Por ejemplo, para crear una clave llamada cliente con un valor de Pepe, es necesario ejecutar desde la CLI de Redis:

```
> SET idcliente Pepe
```

Observa que las comillas dobles no son necesarias en este caso porque el valor es una única palabra, *Pepe*. Esta instrucción ha creado un valor de cadena simple Pepe que se ha almacenado en la base de datos.

Lo siguiente es recuperarlo con GET:

```
> GET idcliente  
"Pepe"
```

Otra forma de provechar los valores es mediante el uso de contadores. En estos casos, el comando **INCR** (abreviatura de *incremento*) permite incrementar el valor de una clave.

```
> SET sesioncount 1  
> INCR sesioncount  
(entero) 2  
> GET logincount  
"2"
```

No se pueden usar comandos destinados a datos numéricos en datos de cadenas.

Los *bitmaps*, por su parte, son una forma de almacenamiento de cadenas donde se pueden representar muchos elementos de datos que posean, por ejemplo, un estado determinado activado (1) o desactivado (0) (el ejemplo más común es un usuario que está activo o inactivo).

En un escenario donde solo se quiere saber estos estados, Redis es útil por el rápido acceso y manipulación de dichos valores. Debido a que solo puede ser uno de dos valores, puede representar esos datos de manera eficiente. 512 MB es el tamaño máximo de un valor de cadena (según la versión de Redis), lo cual permite almacenar 232 valores posibles en dicha cadena.

SETBIT y **GETBIT** son comandos específicos para trabajar con tipos *bitmaps*. Se utilizan para crear o cambiar un valor y recuperar un valor, respectivamente. Otros comandos son **BITOP** y **BITFIELD**.

Lists

En determinados escenarios una lista puede ser una matriz, pero Redis la utiliza como lista vinculada que permite operaciones de escrituras muy rápidas. Se usan para almacenar datos con algún tipo de relación. En el caso de las operaciones de lectura, estas pueden verse afectadas por la posición donde se encuentre el elemento en la lista.

Aunque no siempre es la mejor opción debido a los valores repetidos, cuando las operaciones de lectura son importantes, se puede usar un **SET**. *List* implementa una clave que contiene varios valores ordenados como *string*.

Los valores se pueden añadir al principio (*left*) o al final (*right*) de la lista y recuperar dichos valores por su índice. Como existe libertad para almacenar valores, estos podrán repetirse y hacer uso de claves diferentes para su consulta.

L PUSH y **R PUSH** permiten la inserción de un valor en una lista. El nuevo valor se posiciona bien al principio de la lista o bien al final respectivamente.

```
>L PUSH cliente Pepe Javier Juan
```

La lista **cliente** almacena tres elementos, indexados de 0 a 2. Como en toda lista, es posible recuperar cada elemento mediante **L INDEX**.

```
>L INDEX cliente 0
"Juan"
>L INDEX cliente 1
"Javier"
>L INDEX cliente 2
```

Observa el orden de los elementos cuando fueron insertados y el orden que les corresponde al consultarlos.

Supongo que el lector de este tema podrá imaginar qué ocurrirá si se intenta acceder a una posición de la lista que no existe. En el caso de querer recuperar un rango de valores, con el comando **LRANGE** es posible, indicando como parámetro la primera y última posición.

```
> LRANGE cliente 0 -1
"Juan"
"Javie"
"Pepe"
(-1) significa hasta el final de la lista.
```

Sets

Los *sets* son parecidos a las *lists*, pero se diferencian en dos aspectos:

1. En la forma de almacenar los valores porque utiliza una sola clave para almacenar varios valores.
2. En la forma de recuperar los valores; no se usa índice de posición y tampoco se ordenan.

A diferencia de las listas, los conjuntos no pueden tener miembros repetidos dentro de la misma clave. Para evitarlo, Redis gestiona el almacenamiento interno de los *sets*. Los *sets* tampoco tienen la operativa de asignar los valores *left* o *right*. Para añadir valores se utiliza el comando **SADD**. Consultar todos los elementos de un *set* es posible gracias a al comando **SMEMBERS**.

```
> SADD sesion user1
> SMEMBERS session
```

1) “user1”

También es posible comprobar si existe un elemento dentro de *set* empleando el comando `SISMEMBER`. Cuando el elemento existe, se devuelve el entero 1. Si el elemento no existe, devuelve un entero 0.

```
> SISMEMBER session user1  
(integer) 1
```

Hashes

Se emplean para almacenar colecciones de pares clave-valor. Un *hash* tiene una clave, pero dentro de su estructura hay espacio para más campos y valores. El estado actual de una aplicación es algo que se suele almacenar en un *hash*.

El comando para crear el *hash* es `HSET`. Este permite recuperar valores concretos en un *hash* concreto.

```
> HSET applicationID session 12 requests 123 forms "register and login"  
> HGET applicationID session  
12
```

Sets ordenados

Una variante de *sets* es su versión ordenada de los valores. Este tipo de datos se utiliza cuando se quiere almacenar datos que deben estar clasificados. Al igual que *hash*, una sola clave puede almacenar varios elementos. Cada valor está puntuado con un número. Un ejemplo para este caso podría ser las veces que determinados usuarios retuitean durante el día:

- ▶ Pepe: 51.
- ▶ Ana: 18.
- ▶ Farith: 33.

```
> ZADD usuariosRetweet 51 Pepe 18 Ana 33 Farith
```

Si se quiere conocer el *set* y sus valores, se emplea el comando **ZRANGE**. Similar a **LRANGE**, existe el comando **ZRANGE** que permite recuperar los valores de una lista.

```
> ZRANGE usuariosRetweet 0 -1
```

ZRANGE recupera los miembros, pero no sus valores (el número de retuits). Para recuperar tanto los nombres de los elementos como sus retuits, se añade el parámetro **WITHSCORES**.

```
> ZRANGE usuariosRetweet 0 -1 WITHSCORES
```

- 1) "Ana"
 - 2) "18"
 - 3) "Farith"
 - 4) "33"
 - 5) "Pepe"
 - 6) "51"
-

Observa que los elementos se ordenan de menor a mayor por el número de retuits.

Una forma de consultarlos con el orden contrario es mediante el comando **ZREVRANGE**.

```
> ZREVRANGE usuariosRetweet 0 -1 WITHSCORES
```

Los elementos se pueden manipular de forma individual. El comando **ZINCRBY** permite incrementar cualquier número. Para incrementar el número de retuits del usuario Pepe en 11, se emplea el siguiente comando:

```
> ZINCRBY usuariosRetweet 11 Pepe  
"62"
```

Con el comando `ZRANK` es posible determinar en qué parte del *set* se encuentra un elemento dado.

HyperLogLog

Este es un tipo de datos especializado muy útil en Redis. Se utiliza para mantener un recuento estimado de elementos únicos. Un caso de uso común es emplearlo para llevar a cabo un seguimiento de un recuento general de visitantes únicos al aula virtual.

Este tipo de datos utiliza *hash* interno para determinar si ya existe el valor.

Si el valor existe, este no se guarda; en caso contrario, sí se almacena. `PFADD` se usa tanto para crear una clave como para añadir nuevos elementos a una clave en este tipo de datos.

> `PFADD estudiante 178.263.1.253`

Si la IP 178.263.1.253 es la primera vez que visita el aula virtual, el comando retorna 1 para indicar que se ha añadido la nueva IP. En caso contrario, retorna 0 si la IP ya existe. `PFCOUNT` permite estimar el número de elementos únicos dentro de un *HyperLogLog*.

Estructuras

Una funcionalidad interesante de Redis es el patrón *publisher/subscriber* (pub/sub) el cual actúa como un mecanismo para intercambiar mensajes. Al usarlo, un publicador crea un par clave-valor y otros clientes se suscriben a él para recibir dicho mensaje (el par clave-valor creado).

Para lograr esta comunicación, se usa el comando **PUBLISH** para crear el valor y publicarlo.

```
> PUBLISH clima temperatura:32c
```

El mensaje **clima** se publica en el canal independientemente de si hay o no clientes suscritos. El cliente que esté suscrito recibirá un mensaje como el siguiente:

-
- 1) "mensaje"
 - 2) "clima"
 - 3) "temperatura:32c"
-

Un cliente podrá suscribirse a determinado canal utilizando el comando **SUBSCRIBE**. Es importante que el cliente sepa qué formato tiene el mensaje para poder procesarlo. Los canales pub/sub se pueden dividir para crear una estructura jerárquica por convención o personalización.

Supón que quieras publicar sobre el clima, en concreto, la temperatura de determinada zona de Madrid.

```
> PUBLISH clima:Madrid temperatura:33c
```

Un cliente podrá suscribirse a la Comunidad de Madrid para recibir los datos del clima. En el caso de que el cliente quiera todas las subclaves, es posible usar el comodín (*).

```
> PSUBSCRIBE clima:*
```

Geoespacial

La indexación geoespacial es un patrón común utilizado para codificar datos basados en la latitud y la longitud, la cual facilita y agiliza el trabajo con datos espaciales.

Cuando los datos están almacenados, es posible calcular la distancia entre dos puntos (datos) utilizando funciones integradas.

```
> GEOADD torreMadrid -31.278 33.589 picaso
> GEOADD torreMadrid -44.155 22.894 bankia
> GEODIST torreMadrid pisaco bankia
> GEODIST torreMadrid pisaco bankia mi
```

En la última instrucción se calcula la distancia entre las dos torres de Madrid usando el comando **GEODIST**. Por defecto este comando devuelve valores en metros, si se quiere utilizar, por ejemplo, millas, se añade al final **mi**.

Streams

En esta estructura, *stream* hace referencia a un patrón de transmisión mucho más potente que pub/sub. Con PUBLISH no se almacenan los datos que se publican. En el caso de *stream*, los consumidores crean un identificador único para gestionarlo ellos mismos.

Como puede que los emisores o publicadores almacenen datos antiguos, los nuevos consumidores podrán en cualquier momento pedir todos los mensajes disponibles.

Otra característica importante radica en que los mensajes se pueden etiquetar como leídos por un cliente determinado. El comando **XADD** crea un *stream* y otros comandos como **XRANGE** permiten manipular dicho elemento.

Al solicitar los mensajes, se pueden pedir solo aquellos que estén pendientes y, en base a esos, realizar otras acciones.

Módulos

Módulos principales	
Redis Graph	Implementa una base de datos de grafos en Redis. Las bases de datos de grafos proporcionan un método para implementar la teoría de grafos sobre los datos relacionados.
RedisSearch	Es un motor de búsqueda de texto completo basado en el almacenamiento de documentos dentro de Redis. Este módulo utiliza funciones de búsqueda de alto rendimiento. Dentro de sus características está el uso de búsquedas ponderadas, la lógica booleana y el autocompletar, entre otras.
RedisTimeSeries	Permite almacenar datos de series de tiempo y trabajar con dichos datos de una forma rápida.

Tabla 1. Módulos implementados por Redis para datos específicos.

9.4. Arquitectura y topología

Los entornos productivos se caracterizan por su nivel de rendimiento y redundancia (Carlson, 2013). Para conseguir el rendimiento de una base de datos, se pueden emplear varios mecanismos, entre ellos, la agrupación en clústeres y la fragmentación (Paksula, 2010). Un fragmento de base de datos es una pequeña porción de una base de datos mucho más grande. Estas partes de la gran base de datos se ubican entre diferentes servidores para que cada uno se haga responsable de un subconjunto de los datos.

La versión Enterprise de Redis puede realizar agrupamiento en clúster integrados. Esas partes de la base de datos de Redis se comparten también en un conjunto de servidores. Siguiendo la descripción anterior, cada servidor dentro de un clúster concreto es responsable solo de su propio conjunto de datos.

Redis separa la administración del clúster de la arquitectura para que las solicitudes sean atendidas tan rápido como lo serían si el servidor no se estuviera ejecutando en un

clúster (Da Silva y Tavares, 2015). Un servidor dentro de un clúster se denomina nodo en Redis. Los nodos podrán ser primarios o secundarios.

El clúster de Redis Enterprise consta de los siguientes componentes:

- ▶ **Capa de datos:** en esta capa los datos se almacenan y administran, haciendo parte del mismo núcleo como si se tratase de una única instancia de Redis (*open source*).
- ▶ **Administrador de clústeres:** se encarga del estado general del clúster, su supervisión, el reequilibrio, la compartición, el aprovisionamiento y el desaprovisionamiento de nodos, entre otras tareas.
- ▶ **Proxy de latencia cero:** los nodos del clúster utilizan un *proxy* para proporcionar comunicación sin estado y multiproceso entre el cliente y el nodo.
- ▶ **API REST:** se utiliza para la gestión del clúster.

Alta disponibilidad

Es importante en este punto volver a diferenciar las dos versiones que ofrece Redis para trabajar. Por un lado, está Redis Open Source y, por otra parte, está Redis Enterprise. Cuando se quiere conseguir una alta disponibilidad, con Redis Open Source el gasto en RAM dificulta las réplicas del dato de forma simultánea.

Recordemos que, para proporcionar una alta disponibilidad en el caso de divisiones de red, es necesario ejecutar tres réplicas de los mismos datos simultáneamente (Choi et al., 2016). En caso de un fallo en la red, los dos nodos restantes que pueden comunicarse pasan a ser autorizados.

Redis Enterprise, por su parte, ofrece alta disponibilidad sin necesidad de una tercera réplica. En lugar de ello, emplea un tercer servidor mucho más pequeño para la resolución del *quorum* en el caso de divisiones de red. Esta manera evita la necesidad de

una RAM costosa, lo que, en cualquier escenario, significa ahorros de costos directos. La replicación en RAM se lleva a cabo entre maestro y esclavo.

Otra ventaja es que monitorea tanto a nivel de nodo como a nivel de clúster, garantizando que los procesos relacionados con el rendimiento de los nodos funcionen correctamente. Si un nodo deja de estar disponible o no responde, el guardián del nodo comienza el proceso de comutación por error del fragmento.

9.5. Despliegue en Docker

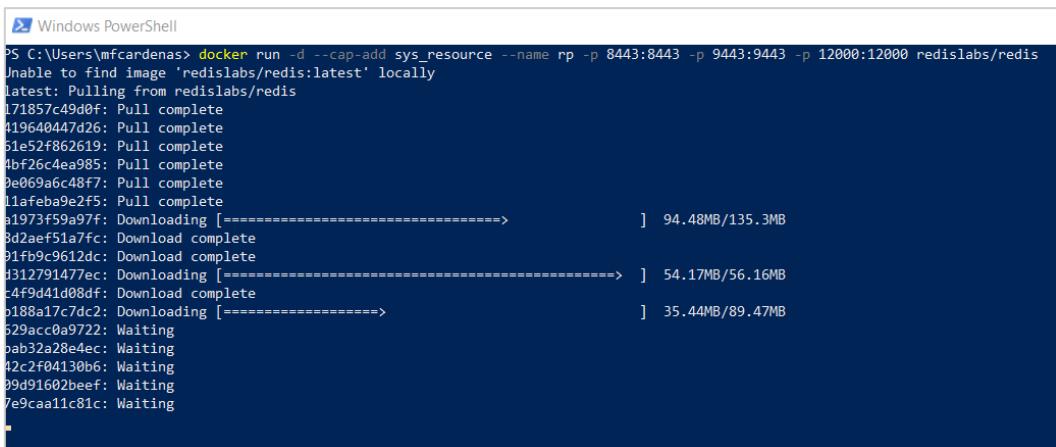
Para que puedas poner en marcha un servidor de Redis, en este apartado se propone una instalación basada en Docker que permita de forma rápida acceder a Redis y realizar las pruebas de los comandos antes vistos. A continuación, se indican los pasos de instalación.

- ▶ **Paso 1:** inicia Docker Desktop.
- ▶ **Paso 2:** ejecuta la instrucción de descarga y configuración de imagen de Redis.

```
docker run -d --cap-add sys_resource --name rp -p 8443:8443 -p 9443:9443 -p 12000:12000 redislabs/redis
```

Docker con RS se ejecuta en su *host* local con el puerto 8443, el cual está abierto para conexiones HTTPS, 9443 para conexiones de API REST y el puerto 12 000 abierto para conexiones de cliente de Redis.

Si lo deseas, puedes utilizar otros puertos con `-p <host_port>: <container_port>` o usar la opción `--network host` para abrir todos los puertos a la red *host*.



```

PS C:\Users\mfcardenas> docker run -d --cap-add sys_resource --name rp -p 8443:8443 -p 9443:9443 -p 12000:12000 redislabs/redis
Unable to find image 'redislabs/redis:latest' locally
latest: Pulling from redislabs/redis
171857c49d0f: Pull complete
419640447d26: Pull complete
61e52f862619: Pull complete
4bf26c4ea985: Pull complete
9e069a6c48f7: Pull complete
11afeba9e2f5: Pull complete
a1973f59a97f: Downloading [=====] 94.48MB/135.3MB
3d2aef51a7fc: Download complete
91fb9c9612dc: Download complete
312791477ec: Downloading [=====] 54.17MB/56.16MB
c4fd41d08df: Download complete
9188a17c7dc2: Downloading [=====] 35.44MB/89.47MB
529acc0a9722: Waiting
bab32a28e4ec: Waiting
42c2f04130b6: Waiting
39d91602beef: Waiting
7e9caa11c81c: Waiting

```

Figura 2. Ejecución de un contenedor desde Docker hub.

Si revisas Docker Desktop, deberías poder ver la nueva imagen de Redis en ejecución.

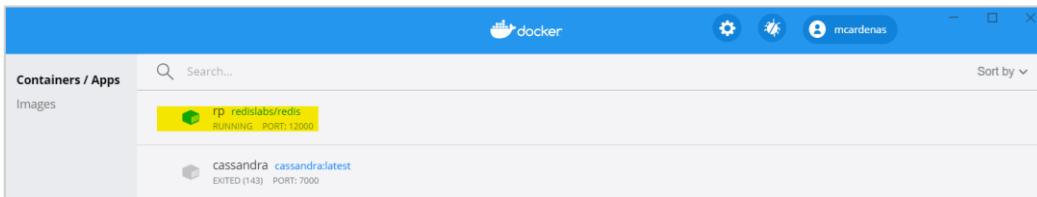


Figura 3. Ejecución de un contenedor desde Docker hub.

- **Paso 3:** inicia la aplicación web del servidor de Redis en la siguiente URL: <https://localhost:8443> y da clic en el botón Setup para configurar tu servidor local.

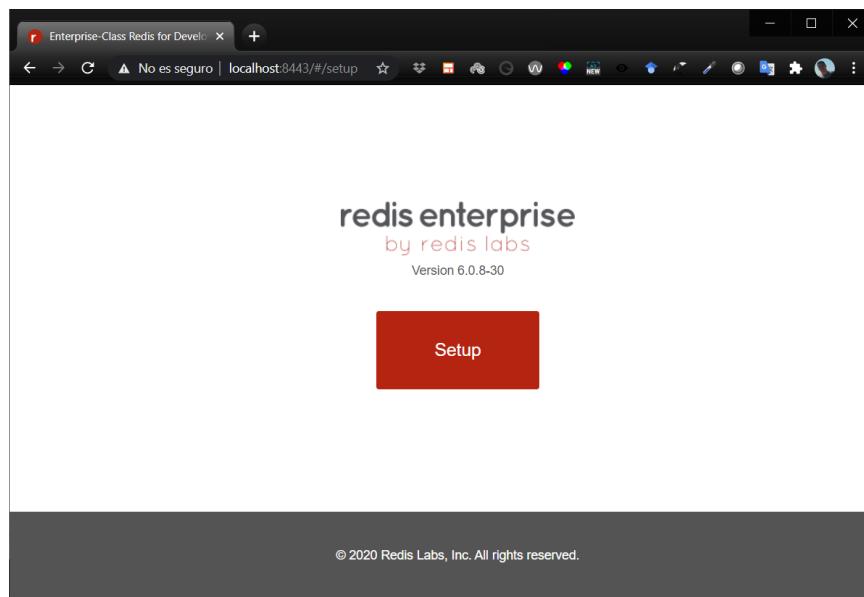


Figura 4. Web principal de Redis.

- ▶ **Paso 4:** asigna un nombre (FQDN) a tu clúster: «clusterlocal», por ejemplo, y luego da clic en el botón Next.

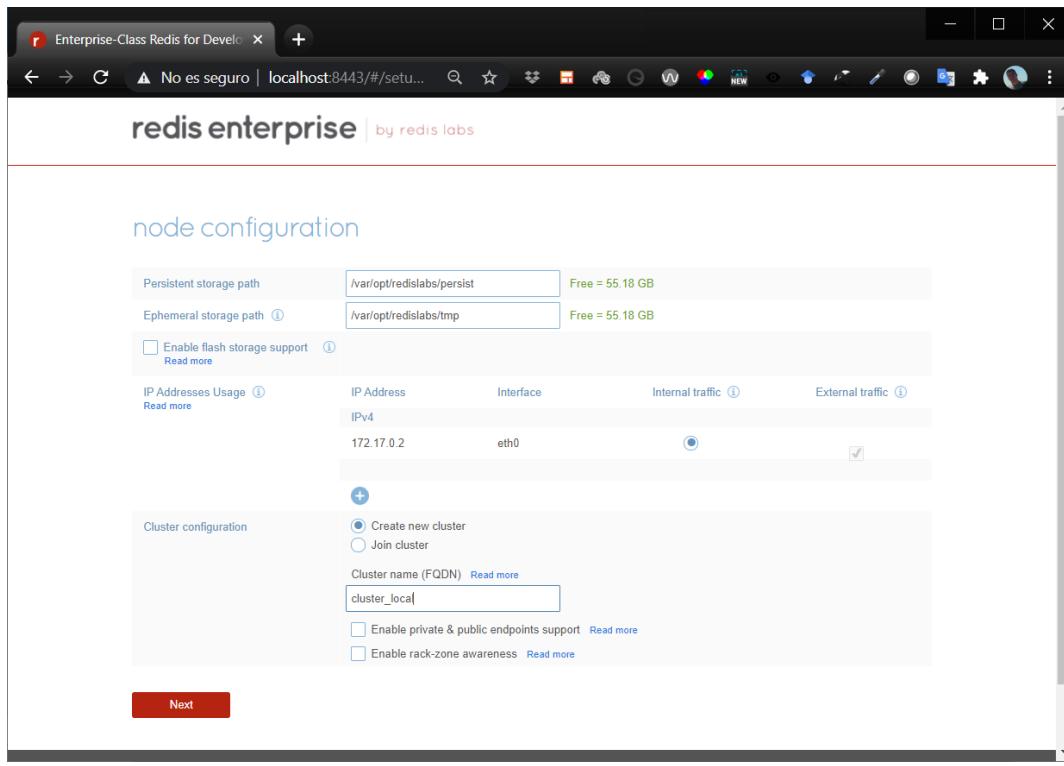


Figura 5. Configuración y creación de base de datos en Redis.

- ▶ **Paso 5:** cuando solicite la licencia omite introducirla y da clic en Next. Luego introduce un correo y asigne una contraseña a su cuenta de administrador.

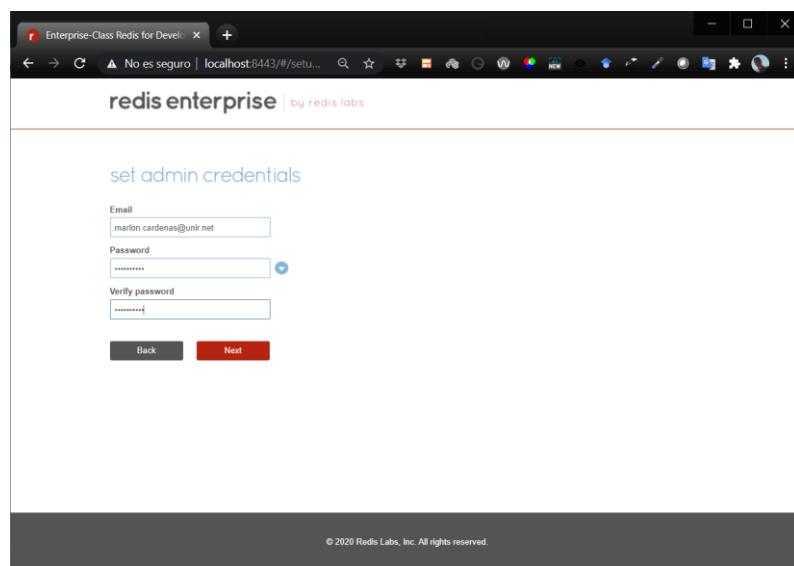


Figura 6. Creación de cuenta administrador de Redis.

- **Paso 6:** marca la opción Crear base de datos y haz clic en el botón Next.

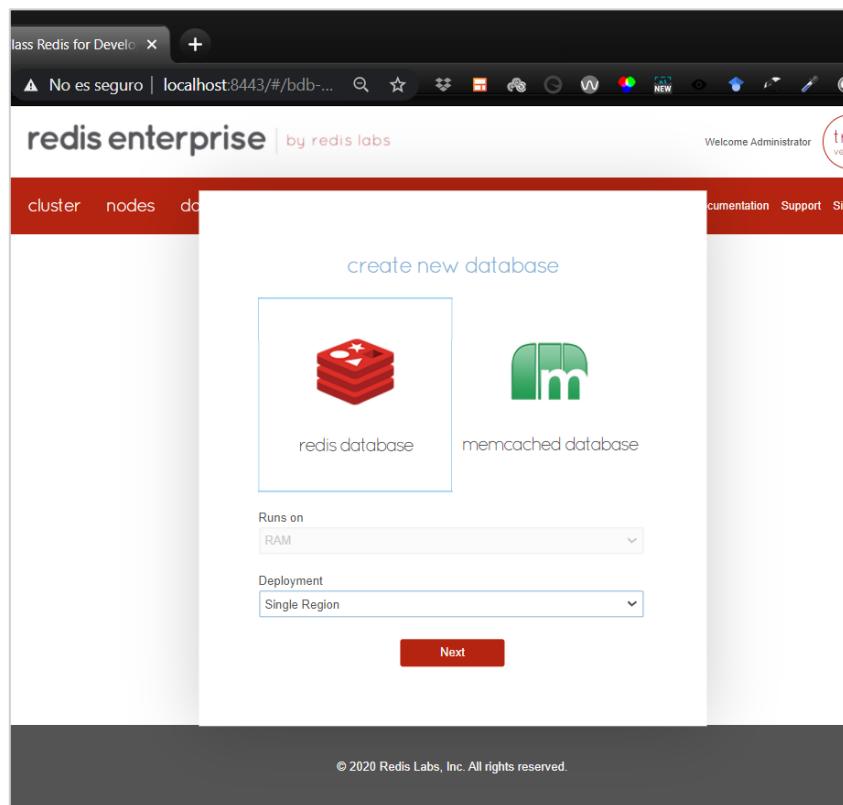


Figura 7. Selección del tipo de base de datos, base de datos normal o en memoria.

- **Paso 7:** asigna como nombre «databasetest», luego haz clic en Show advance options y en el campo Endpoint port number indica el puerto 12 000.

Si el puerto 12 000 no está disponible, ingresa cualquier puerto disponible entre 10 000 y 19 999 y conéctate a la base de datos con ese número de puerto.

Observa cómo cada base de datos utiliza un puerto por defecto en las instalaciones por defecto: MySQL utiliza el puerto 3306, MongoDB el puerto 27017, Neo4J el puerto 7474... Por seguridad, los administradores de bases de datos cambiarán puertos menos comunes.

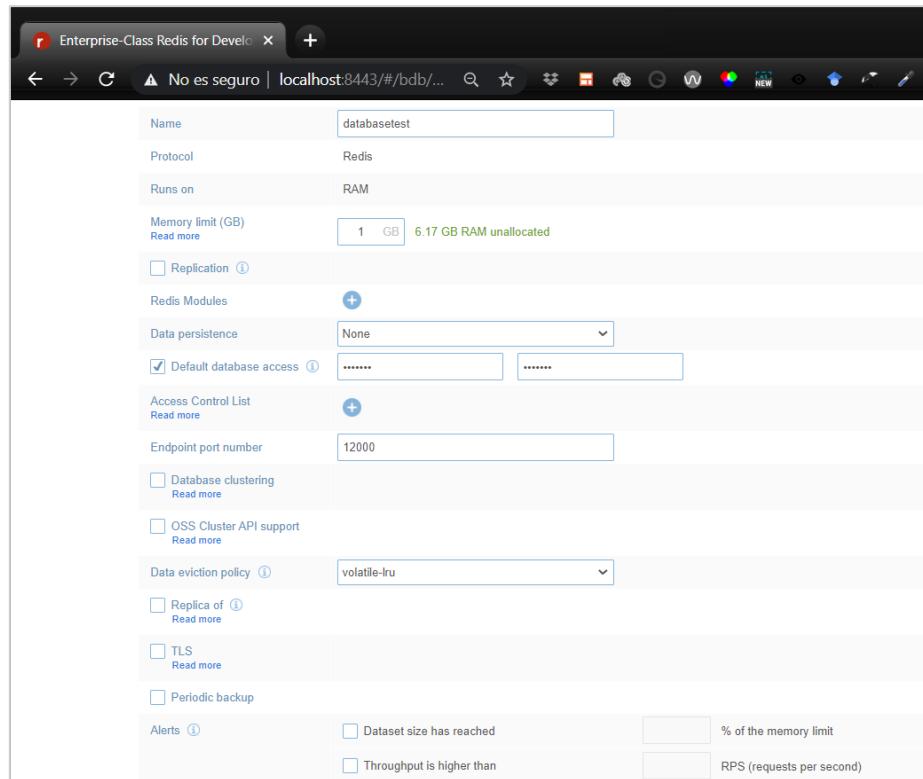


Figura 8. Configuración de la base de datos en Redis.

Después de crear la base de datos en Redis, la página web que debes ver es la siguiente.

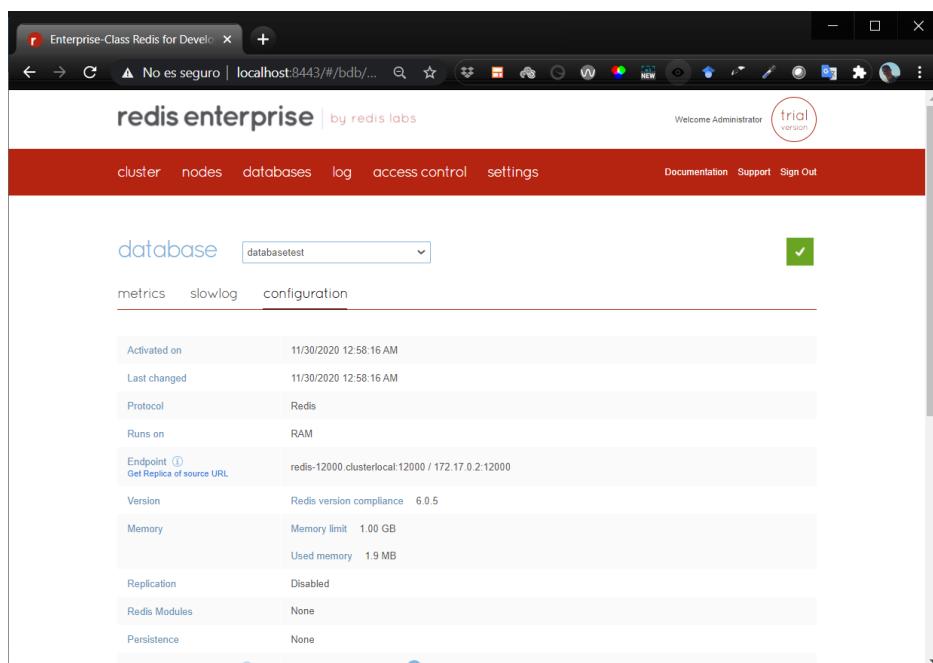
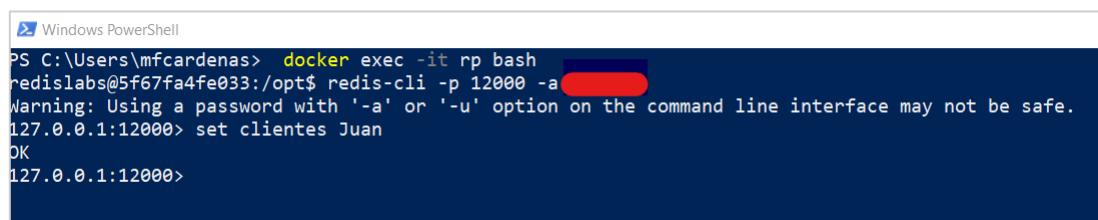


Figura 9. Configuración avanzada de la base de datos en Redis.

Si no puedes activar la base de datos debido a una limitación de memoria,
asegúrate de que Docker tenga suficiente memoria asignada.

- ▶ **Paso 8:** inicia redis-cli, conéctate a la base de datos creada y practica con todos los comandos vistos en este tema.

```
> docker exec -it rp Bash  
> redis-cli -p 12000 -a supasswordsilopuso
```



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command entered is "redislabs@5f67fa4fe033:/opt\$ redis-cli -p 12000 -a [REDACTED]". A warning message follows: "Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe." Then, the command "set clientes Juan" is run, followed by "OK" and the IP address "127.0.0.1:12000>".

Figura 10. Consola Bash de Redis desde un contenedor en Docker hub.

9.6. Drivers

Redis se integra con muchos lenguajes de programación conocidos. De forma programática, desde estos lenguajes, es necesario crear los mecanismos para realizar dicha conexión. Los clientes y controladores generalmente se comparten bajo una licencia de código abierto, aunque la licencia varía según el proyecto.

Consulta <https://redis.io/clients> para obtener más información sobre clientes de Redis.

A continuación, una lista de lenguajes compatibles con Redis:

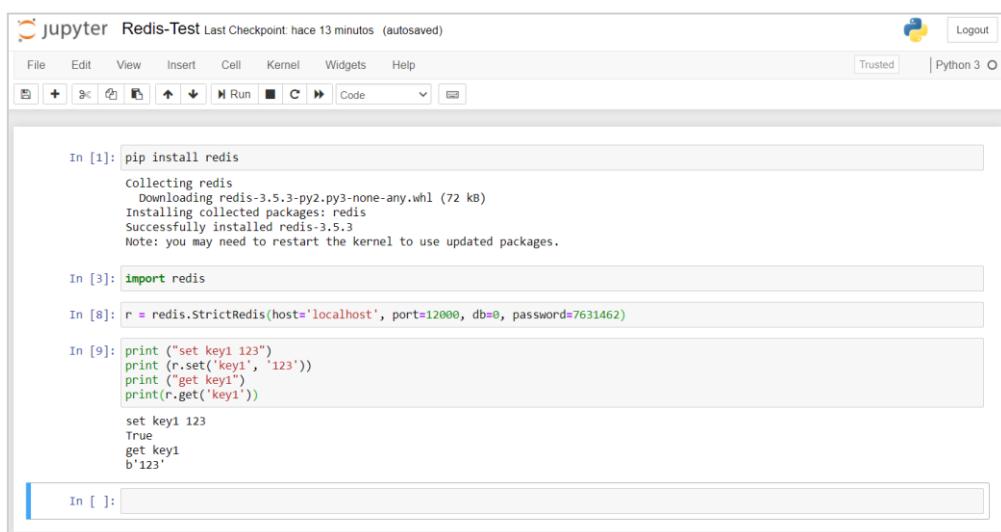
- ▶ **Python:** se integra con Redis a través del paquete redis-py, aunque también existen otros paquetes igual de compatibles.
- ▶ **Java:** mediante estos tres clientes: Jedis, Lettuce y Redisson.

- ▶ **Node.js**: el cliente recomendado para Node.js es node_redis.
- ▶ **C#**: dos clientes populares son ServiceStack.Redis y StackExchange.Redis.
- ▶ **PHP**: tiene varios clientes con PHP, pero el más recomendado es Predis.
- ▶ **C**: el cliente oficial de Redis para el lenguaje C es hiredis. Por otro lado, también está hiredis-vip, el cual permite obtener información sobre compatibilidad con el lenguaje C relacionado con el clúster.

Usando Python

A continuación, se indican los pasos para usar Python con la instalación de Redis llevada a cabo en el apartado anterior. Sigue las siguientes instrucciones desde Jupyter de Anaconda u otra instalación similar.

```
pip install redis
import redis
r      =      redis.StrictRedis(host='localhost',      port=12000,      db=0,
password=supassword)
# instrucciones de ejemplo
print ("set key1 123")
print (r.set('key1', '123'))
print ("get key1")
print(r.get('key1'))
```



The screenshot shows a Jupyter Notebook interface with the title "Redis-Test". The notebook has a toolbar with File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Python 3 kernel selector. Below the toolbar, there are buttons for Run, Cell, Kernel, Help, and a Trusted status indicator. The notebook contains the following code:

```
In [1]: pip install redis
Collecting redis
  Downloading redis-3.5.3-py2.py3-none-any.whl (72 kB)
Installing collected packages: redis
Successfully installed redis-3.5.3
Note: you may need to restart the kernel to use updated packages.

In [3]: import redis
In [8]: r = redis.StrictRedis(host='localhost', port=12000, db=0, password=7631462)

In [9]: print ("set key1 123")
print (r.set('key1', '123'))
print ("get key1")
print(r.get('key1'))

set key1 123
True
get key1
b'123'
```

Figura 11. Notebook Python con instrucciones de la práctica.

Después de leer todo el tema, en la lección «Características principales de Redis» podrás repasar los conceptos generales de la base de datos y revisar los tipos de datos que maneja Redis. De igual forma, podrás ver una breve descripción del uso de Redis en determinados casos de uso.



Vídeo 1. Características principales de Redis.

Accede al vídeo a través del aula virtual

9.7. Referencias bibliográficas

Carlson, J. L. (2013). *Redis in action*. Manning Publications.

Choi, J. M., Jeong, D. W., Yoon, J. S., y Lee, S. J. (2016). Digital forensics investigation of Redis database. *KIPS Transactions on Computer and Communication Systems*, 5(5), pp. 117-126.

Da Silva, M. D., y Tavares, H. L. (2015). *Redis Essentials*. Packt Publishing.

Paksula, M. (2010). Persisting objects in Redis key-value database. University of Helsinki. <https://www.cs.helsinki.fi/u/paksula/misc/redis.pdf>

A fondo

Documentación oficial: Redis motor NoSQL

Redis Labs. (2021). Get started with Redis [Página web]. <https://redislabs.com/get-started-with-redis/>

Redis es una base de datos que cuenta con una documentación muy detallada y orientada a casos de usos que pueden apoyarse en este tipo de bases de datos. En su página web la documentación de la herramienta se enfoca en tres aspectos importantes: configuraciones de uso, conexión a la base de datos y uso de clientes para explotar Redis.

Documentación oficial: Redis en su versión Enterprise, mejoras funcionales

Redis Labs. (2021). Documentation [Página web].
<https://docs.redislabs.com/latest/index.html>

El equipo de Redis dispone de la documentación de la versión Enterprise de esta herramienta. Aunque es una versión de pago, es interesante que el estudiante revise dicha documentación para que conozca las funcionalidades que cubre dicha versión, con respecto a la versión *open source*.

Fundamentos de Redis: *Redis Crash Course*

Traversy Media. (19 de marzo de 2017). *Redis Crash Course Tutorial* [Archivo de vídeo].
<https://www.youtube.com/watch?v=Hbt56gFj998>



En el siguiente vídeo se lleva a cabo una presentación de los fundamentos de Redis. En el vídeo se discuten y demuestran las principales características de la base de datos, incluyendo su definición e instalación, el cliente redis-cli, los tipos de datos, cadenas, listas, conjuntos, conjuntos ordenados, *hash* y la persistencia de datos.

Ejemplo de soluciones con Redis: casos de uso comerciales

Redis Labs. (2021). Case Studies [Página web]. <https://redislabs.com/case-studies/>

La mejor manera de comprender el uso que se le puede dar a las bases de datos como Redis es analizando casos de estudios donde actualmente se esté implementando. La web oficial de Redis dispone de varios casos de uso que pueden ser estudiados para aprender el dominio de este tipo de base de datos. Consulta estos casos de uso, pero sobre todo céntrate en descubrir en qué escenario se está implementado Redis y de qué forma cubre la problemática o necesidad planteada.

- 1.** Las siguientes son estructuras de Redis:
 - A. *String, integer, float, sets y lists.*
 - B. *Lists, sets, hashes, stream y bit arrays.*
 - C. *HyperLogsLogs, hashes, sets, string y float.*
 - D. Ninguna de las anteriores es correcta.

Estos son algunos de los tipos de datos que utiliza Redis para almacenar y manipular los datos. *Float*, por su parte, no es un tipo de datos de Redis.

- 2.** ¿Cuál de estas afirmaciones es verdadera?
 - A. No es necesario definir previamente un modelo de datos antes de usar o guardar datos en Redis.
 - B. Redis se caracteriza por el tratamiento de distintos tipos de datos, los cuales se pueden aplicar en distintos casos de uso.
 - C. Redis es una base de datos clave-valor.
 - D. Todas las anteriores son correctas.

Todas las afirmaciones son ciertas, hacen parte de las características relevantes de la base de datos.

- 3.** ¿Cómo se llama el cliente de Redis?
 - A. Cliente Redis Long.
 - B. redis-cli.
 - C. cli-rediss.
 - D. Ninguna de las anteriores es correcta.

Redis llama a su consola o terminal *redis-cli*, abreviatura de *cliente redis* en inglés.

4. ¿Cuál de estas instrucciones muestra todos los elementos de una lista llamada clientes?

- A. LINDEX clientes 0.
- B. LRANGE clientes 0 -1.
- C. LRANGE clientes 0 *.
- D. GET clientes 1.

Los valores 0 y -1 indican que se muestren los valores desde la posición 0 hasta el final de la lista.

5. ¿Cuál de estas instrucciones permite añadir un elemento a la colección llamada frutas?

- A. SADD frutas mango.
- B. SADD mango frutas.
- C. SMEMBERS frutas mango.
- D. SISMEMBERS frutas mango.

SADD permite añadir elementos a una colección, seguido se indica el nombre de la colección y luego el elemento a añadir.

6. ¿Qué hace la siguiente instrucción: SET user pep?

- A. Establece un usuario en pep.
- B. Crea un elemento llamado user con el valor pep.
- C. Muestra el valor de user y pep.
- D. Guarda el valor user en pep.

SET crea un elemento, seguido se indica el nombre del elemento y luego su valor.

7. ¿Qué son las colecciones ordenadas?

- A. Una colección cuyos elementos se ordenan por la clave.
- B. Una colección cuyos elementos se ordenan por el orden en que se añaden los valores.
- C. Una colección cuyos elementos se ordenan por los valores.
- D. Ninguna de las anteriores es correcta.

Las colecciones ordenadas se ordenan por los valores que se añaden a dicha colección, no por su clave.

8. El patrón pub/sub permite:

- A. Publicar datos cada cierto tiempo en un puerto concreto.
- B. Crea un canal para que los clientes se suscriban y reciban los valores emitidos.
- C. Publicar datos en *stream* para ser consumido.
- D. Todas las anteriores son correctas.

Según la configuración que se le aplique al patrón, este permitirá hacer las tres operaciones que se indican previamente.

9. Redis es una base de datos clave-valor que utiliza:

- A. RAM y memoria *flash*.
- B. Solo RAM.
- C. RAM y memoria caché.
- D. RAM SSD y Sweep.

Redis posee funciones especializadas para usar tanto la memoria RAM como las memorias *flash*, incluso en SSD.

10. ¿Qué afirmación es verdadera?

- A. Dentro de un clúster de Redis, un servidor determinado se denomina *cliente*.
- B. Cada nodo puede ser un nodo primario (maestro), pero no un nodo secundario (esclavo).
- C. La administración del clúster se realiza en una capa de la arquitectura del clúster de Redis.
- D. El almacenamiento de datos de series de tiempo es otra tarea común en Redis.

Redis dispone de un módulo que permite tratar datos de series de tiempo.

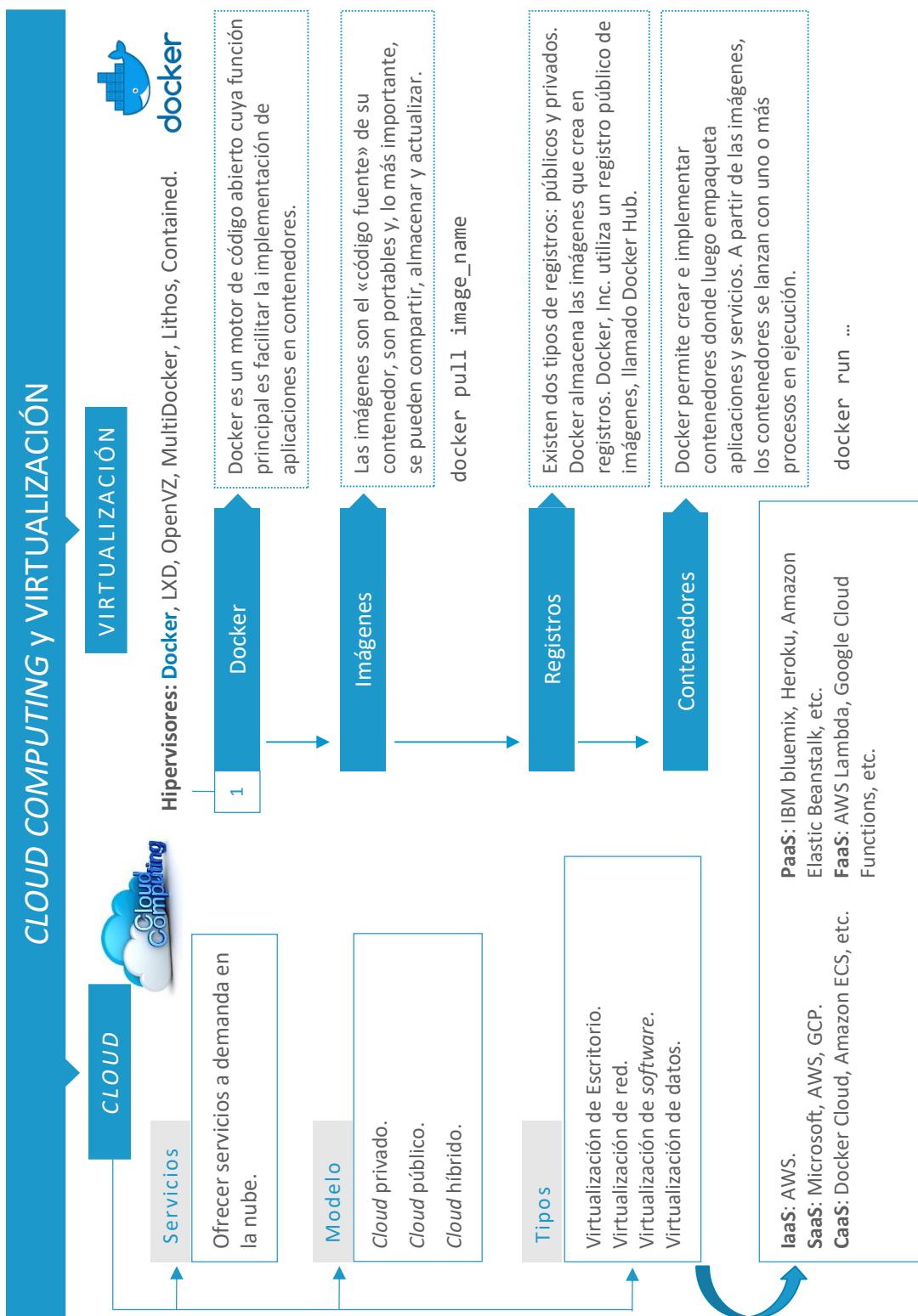
Bases de Datos para el Big Data

Datos en el *cloud*

Índice

Esquema	3
Ideas clave	4
10.1. Introducción y objetivos	4
10.2. <i>Cloud computing</i>	5
10.3. Modelos de <i>cloud computing</i>	6
10.4. Tipos de servicios <i>cloud</i>	9
10.5. Virtualización	13
10.6. Docker	16
10.7. Manos a la obra con Docker	21
10.8. Iniciar un contenedor con MongoDB	23
10.9. Referencias bibliográficas	31
A fondo	32
Test	35

Esquema



10.1. Introducción y objetivos

En este tema se conocerán algunos aspectos claves del uso del *cloud* para la carga y análisis de los datos. En la asignatura hasta el momento se han estado revisando bases de datos concretas aplicadas en el ámbito NoSQL. Las bases de datos vistas no son el único activo utilizado para persistir y analizar la información. El *cloud* es otra alternativa que cubre gran número de soluciones, tanto para las grandes empresas como para las compañías más pequeñas.

Además de conocer la definición del *cloud* como solución tecnológica, se describirán los beneficios que conllevan el uso de la computación en la nube, de persistir los datos y delegar determinada gestión de las bases de datos y de poder desplegar procesos de análisis de forma ágil en entornos productivos.

El uso del *cloud* implica, por un lado, confiar en la infraestructura de otros proveedores y, por otra parte, usar y explotar nuestra información siguiendo determinados protocolos. Existen proveedores que cuentan con sus propias infraestructuras en el *cloud*. En otros casos, algunos dependen de otros proveedores del *cloud* para proporcionar los mismos servicios.

Los objetivos que se abordan en este tema son los siguientes:

- ▶ Entender el concepto de computación en la nube.
- ▶ Conocer los beneficios y desventajas de tratar los datos en el *cloud*.
- ▶ Entender los distintos tipos de servicios que se ofrecen en el *cloud*.
- ▶ Diferenciar servicios como PaaS, IaaS, SaaS y FaaS.
- ▶ Comprender los conceptos de virtualización.
- ▶ Hacer uso de Docker como herramienta de virtualización.

10.2. Cloud computing

Oracle define el *cloud computing* como la capacidad de alquilar sus tecnologías de la información, en lugar de comprarlas. En lugar de invertir mucho en bases de datos, *software* y equipos, las empresas optan por acceder a la potencia de cálculo a través de Internet y pagar por lo que consumen (Oracle, 2021).

Cuando la compañía da el paso de «ir al *cloud*», afronta el reto de llevar todo lo que se considera infraestructura de TI a un entorno ajeno al suyo, a un centro de datos que dirige y gestiona un proveedor *cloud* como Amazon, Azure, Google, Oracle, IBM o Informática, entre otros.

Los proveedores *cloud* asumen el control y la gestión de estas infraestructuras en nombre de sus clientes, y sobre ellas vinculan aplicaciones y entornos de desarrollo para construir nuevas capacidades de negocio y nuevas funcionalidades que les permitan a las empresas asumir los cambios en la demanda del mercado.

Entre los beneficios más destacados que ofrece el *cloud computing* se mencionan los siguientes:

- ▶ **Escalabilidad horizontal y vertical:** cuando aumentan las necesidades de las compañías debido a su propio negocio, el *cloud* se adapta y crece en función de la demanda de recursos.
- ▶ **Migraciones flexibles:** las empresas pueden migrar ciertos recursos hacia o desde la nube, o hacia diferentes proveedores de nube, según lo deseen, o automáticamente para obtener mejores ahorros de costos, un mejor rendimiento o para usar nuevos servicios a medida que surgen.
- ▶ **Infraestructura:** los clientes pueden estimar y calcular los recursos para los distintos elementos de la infraestructura. Aunque suene negativo, los administradores de TI no tienen la tarea de estimar y gestionar directamente los recursos de la compañía.

- ▶ **Resistencia cargas de trabajo:** el servicio en la nube provee el almacenamiento constante de los datos en distintas ubicaciones en todo el mundo.
- ▶ **Pago por uso:** los costes se aplican por unidades concretas de recursos usados, sin recargos y con una facturación de servicios muy transparente.
- ▶ **Protección antidesastres:** el *cloud* elimina la necesidad de invertir en *hardware* para copias de seguridad. Los servidores son globales y la gestión de copias la lleva directamente el proveedor a nivel mundial.
- ▶ **Actualizaciones automáticas, trabajo remoto y aumento de la capacidad de colaboración:** no es necesario llevar a cabo procesos de actualización de *hardware* o *software*. los servicios de cara al usuario operan independientemente de la infraestructura, todo lo que el cliente necesita es un mecanismo de conexión a través de Internet.
- ▶ **Seguridad:** el *cloud* brinda los mejores estándares de seguridad disponibles en el mercado. El cliente, las aplicaciones, los datos y el código están protegidos y monitorizados las 24 horas.

10.3. Modelos de *cloud computing*

Existen tres formas de *cloud computing* según su implementación:

- ▶ **Cloud privado:** como su nombre indica, son servicios *cloud* creados para una organización específica y solo para ella. La organización, al tratarse de un servicio exclusivo, puede personalizar dicho servicio eligiendo, por ejemplo, el *hardware* a utilizar, el *software* y las alternativas de almacenamiento que mejor se adapten al volumen de los datos. Se considera que sigue siendo la figura de centro de datos central para una organización.

La razón por la que este modelo es atractivo es porque las dependencias con proveedores externos son mínimas.

Algunos proveedores que brindan este tipo de *cloud* son:

- OpenStack: <https://www.openstack.org>
- CloudStack: <https://cloudstack.apache.org>
- VMware: <https://cloud.vmware.com>



Figura 1. Esquema simple de un *cloud* privado.

- **Cloud público:** las empresas en este caso acceden a servicios del *cloud* a través de Internet. Estos servicios se ajustan a la demanda del servicio y se miden (a efectos de costes y facturación) con la demanda del servicio por minutos u horas. Las suscripciones son otra forma de prolongar el contrato y los servicios.

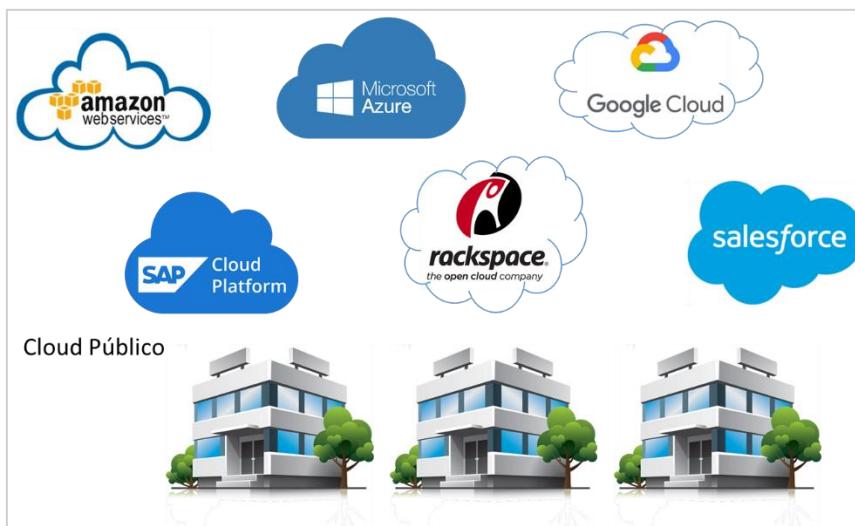


Figura 2. Esquema simple de un *cloud* público.

Cuando se alquila una máquina virtual, por ejemplo, los costes cubren la duración de dicha MV, la capacidad de almacenamiento o el ancho de banda que esta consume. Estas características no solo pueden variar de un proveedor a otro, sino también entre servicios y suscripciones.

Algunos proveedores que brindan este tipo de *cloud* son:

- Microsoft Azure: <https://azure.microsoft.com>
- Google Cloud Platform (GCP): <https://cloud.google.com>
- Servicios web de Amazon: <https://aws.amazon.com>
- IBM: <https://www.ibm.com/cloud>

► **Cloud híbrido:** es una fusión de servicios de *cloud* público (GCP, AWS o Azure) y un *cloud* privado local. Las soluciones en local se usarán más para aplicaciones críticas dentro de la compañía, mientras que el *cloud* público abordará las demandas del servicio.

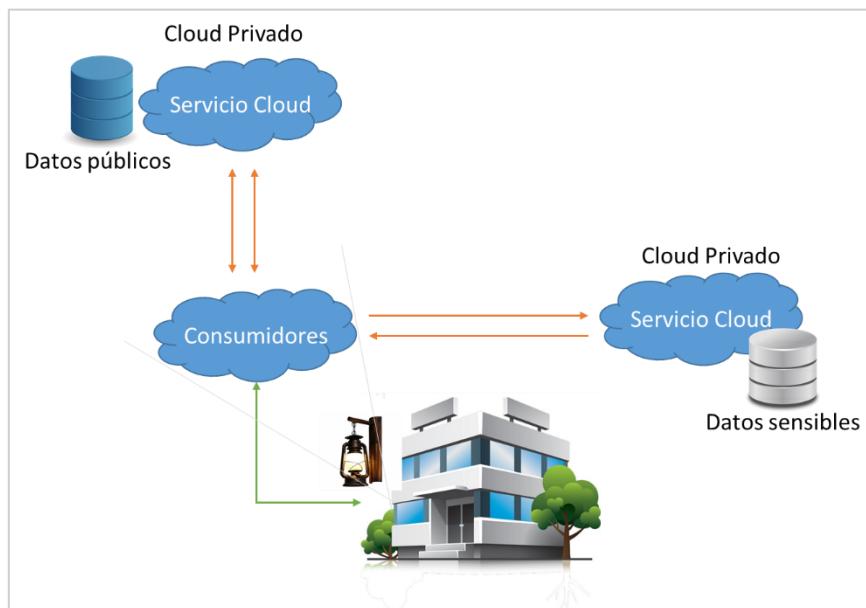


Figura 3. Esquema simple de un *cloud* híbrido.

Algunos proveedores que brindan este tipo de *cloud* son:

- IBM: <https://www.ibm.com/it-infrastructure/z/capabilities/hybrid-cloud>
- AWS: <https://aws.amazon.com/enterprise/hybrid>

Otro tipo de *cloud* que se escucha con más frecuencia es el ***cloud comunitario***, el cual se define como aquel que conforman diferentes empresas u organizaciones con el fin de promover o impulsar un objetivo común.

10.4. Tipos de servicios *cloud*

A continuación, se describen los cinco tipos de servicios que se ofrecen en el *cloud*.

Software as a Service (SaaS)

En este tipo de servicios, el proveedor del *cloud* centra su servicio en alojar e implementar *software* del cliente en su entorno. El cliente accede a sus aplicaciones a través de Internet y se evita así comprar y mantener su propia infraestructura de proceso. El modelo económico empleado es el genérico en las plataformas *cloud*, el cual consiste en el pago por uso de cada tiempo de recurso.

Este tipo de servicios se ajusta a aquellas empresas que demandan acceso rápido a tecnología innovadora. Al delegar al proveedor esta responsabilidad, las actualizaciones automáticas pasan a ser una parte inherente del servicio, reduciendo así la carga de recursos propios de la organización destinados a este tipo de tareas.

El servicio funciona a demanda, si es necesario se amplían o reducen los mismos, según el volumen de trabajo. Algunos productos *software* que se suelen contratar bajo esta tipología son ERP, herramientas para la gestión de cadena de suministro y sistemas contables, entre otros.

Platform as a Service (PaaS)

Este tipo de servicio ofrece acceso a herramientas y plataformas adecuadas para cubrir las necesidades de entornos de desarrollo para crear y gestionar aplicaciones web, móviles o de proceso. De esta forma, se reduce la inversión en la infraestructura que hay alrededor de los equipos de desarrollo internos del cliente.

Por lo general, este tipo de *cloud* provee la infraestructura y *middleware* que luego el cliente consulta a través de portales web.

Infrastructure as a service (IaaS)

También llamado HaaS o Hardware as a Service. En este otro tipo, el servicio se enfoca en ofrecer infraestructura como servicio para que el cliente haga uso del mismo a demanda. Los clientes solo tienen que preocuparse de sus procesos y análisis, delegando a los proveedores del servicio los componentes de infraestructura con capacidad de cómputo, almacenamiento y de red para ejecutar los procesos del cliente.

En IaaS el proveedor dispone de infraestructura para proveer de máquinas virtuales a sus clientes. Con ellas se abstraen los recursos de un ordenador con un *hypervisor* o VMM (Virtual Machine Monitor), el cual crea capas entre máquina *host* y el sistema operativo virtualizado. La virtualización a grandes rasgos permite montar entornos *hardware* mediante *software*.

Es importante destacar que el cliente, en este modelo de negocio, es quien se hace responsable de la instalación, la configuración, la protección de datos y la gestión de las aplicaciones de la infraestructura como bases de datos o *middleware*.

Function as a Service (FaaS)

Otro de los servicios que se ofrecen sobre el *cloud* son funciones como servicio. Este servicio lo utilizan desarrolladores para implementar la lógica de *back-end* sin la necesidad de configurar y levantar un servidor. Dentro de las arquitecturas sin servidor, este servicio está siendo muy utilizado en casi gran parte de las compañías que ofrecen servicio tecnológico en la nube.

Container as a Service (CaaS)

Es un modelo de servicio en la nube que permite a los usuarios cargar, organizar, iniciar, detener, escalar y administrar contenedores, aplicaciones y clústeres. La base de este servicio es la virtualización basada en contenedores, una interfaz de programación de aplicaciones (API) y una interfaz de portal web.

Con CaaS los usuarios pueden construir soluciones sobre contenedores escalables y seguros a través de centros de datos locales o el *cloud*. Los contenedores y clústeres se utilizan como servicio con este modelo y se implementan en la nube o en centros de datos *in situ*.

Enfoque del servicio

Estos servicios brindan varios niveles de abstracción sobre las máquinas que finalmente ejecutan dichos servicios. Cada servicio opera sobre un nivel diferente de responsabilidad dentro del entorno del proveedor.

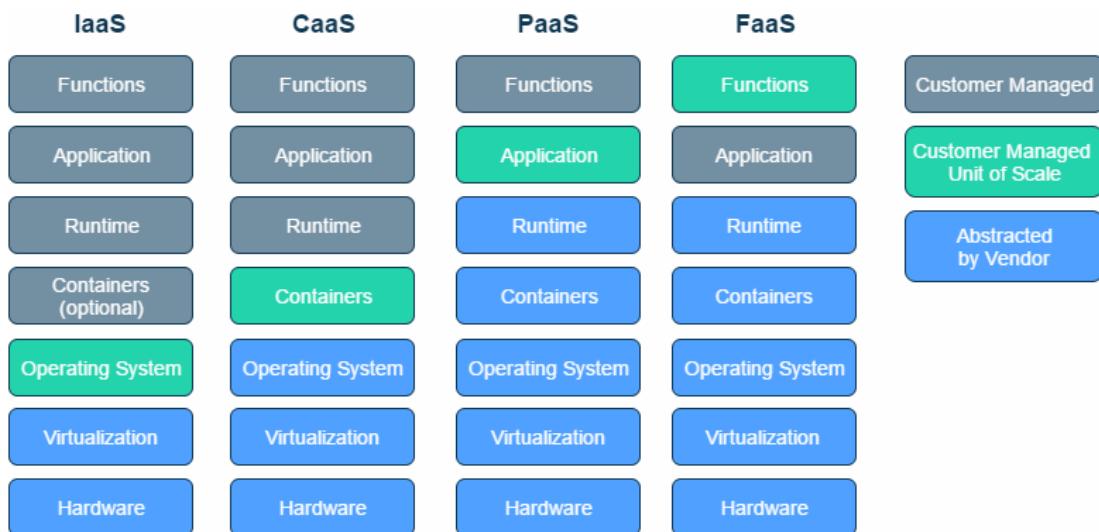


Figura 4. Abstracción de los distintos servicios ofrecidos por el *cloud*. Fuente: <https://serverless.zone/abstracting-the-back-end-with-faas-e5e80e837362>

Actualmente, los desarrolladores pueden optar por implementar código para:

- ▶ IaaS: Amazon EC2, máquinas virtuales de Azure, etc.
- ▶ CaaS: Docker Cloud, Amazon ECS, etc.
- ▶ PaaS: Heroku, Amazon Elastic Beanstalk, etc.
- ▶ Faas: AWS Lambda, Google Cloud Functions, etc.

Cada servicio proporciona un nivel diferente de abstracción y unidad de escala adecuado a sus necesidades. A nivel de aplicaciones, en la Figura 5 se pueden ver cómo distintas tecnologías se enmarcan dentro del tipo de *cloud* ofrecido.



Figura 5. Enfoque de servicio por tipo de *cloud* de IBM. Fuente: [IBM Cloud Strategy for Data-Intensive Enterprises](http://www-03.ibm.com/systems/power/software/big-data/white-papers/Cloud-Strategy-for-Data-Intensive-Enterprises.pdf)

10.5. Virtualización

La virtualización permite crear entornos virtuales simulando un entorno físico. El entorno virtual por lo general incluye características y versiones del *hardware* que representa, de los sistemas operativos que ejecuta, de los dispositivos de almacenamiento configurados, de la memoria RAM empleada, etc.

Con la virtualización es posible convertir un único equipo físico o servidor en varias máquinas virtuales. En él, cada máquina virtual puede interactuar independientemente y levantar sistemas operativos o aplicaciones diferentes mientras comparten los recursos de la máquina *host*.

Lo que se gana con este mecanismo es mejorar la escalabilidad y las cargas de trabajo sobre determinado proceso, al tiempo que permite usar menos servidores reduciendo así el consumo de energía, el mantenimiento y los costes de infraestructura, que pueda acarrear una infraestructura física real.

El uso generalizado de la virtualización redujo la dependencia de un solo proveedor y se transformó en la base del *cloud computing*. Su uso actualmente es tan frecuente que requiere del uso de sistemas de *software* de administración de la virtualización especializados para poder realizar un seguimiento de todas las instancias que puedan crearse o existir.

Los **IaaS** utilizan con frecuencia la virtualización de servidores para ofrecer sus servicios. Mediante un *software* de virtualización crean sobre un servidor físico máquinas virtuales que luego se ofrecen a los clientes para su uso. Los clientes eligen las capacidades de *hardware* que necesitan y otras características como estructuras de red, cortafuegos, copias de seguridad o réplicas, aplicaciones preinstaladas (Spark, Apache, Hadoop, etc.).

The screenshot shows the AWS Free Tier service page. At the top, there's a navigation bar with links like 'Aprender', 'Red de socios', 'AWS Marketplace', 'Capacitación para clientes', 'Más información', and a search bar. Below the navigation is a header for 'Detalles de la capa gratuita' (Free tier details). A sidebar on the left contains filters for 'Tipo de capa' (Tier type) and 'Categorías de productos' (Product categories), with options like 'Destacado', '12 meses de uso gratuito', 'Gratis para siempre', 'Pruebas', 'Análisis', 'Integración de aplicaciones', etc. The main content area displays several service cards:

- COMPUTACIÓN**: Capa gratuita (12 MESES GRATIS) - Amazon EC2: **750 horas** al mes. Description: Capacidad de cómputo de tamaño variable en la nube. Limit: 750 horas por mes de uso de instancias.
- ALMACENAMIENTO**: Capa gratuita (12 MESES GRATIS) - Amazon S3: **5 GB** de almacenamiento estándar. Description: Infraestructura de almacenamiento de objetos segura, duradera y escalable. Limit: 5 GB de almacenamiento estándar.
- BASE DE DATOS**: Capa gratuita (12 MESES GRATIS) - Amazon RDS: **750 horas**. Description: al mes de uso de la base de datos db.t2.micro (se aplica a motores de bases de datos). Limit: 750 horas por mes de uso de bases de datos db.t2.micro (se aplica a motores de bases de datos).
- BASE DE DATOS**: Capa gratuita (GRATUITO PARA SIEMPRE) - Amazon DynamoDB: **25 GB** de almacenamiento. Description: Base de datos NoSQL rápida y flexible con una escalabilidad perfecta. Limit: al mes de uso del cuadro de notas t2.medium durante los primeros dos meses.
- MACHINE LEARNING**: Capa gratuita (PRUEBA GRATUITA) - Amazon SageMaker: **250 horas**. Description: Una plataforma completamente administrada para crear, entrenar e implementar modelos de aprendizaje. Limit: Una plataforma completamente administrada para crear, entrenar e implementar modelos de aprendizaje.

Figura 6. AWS y los servicios que ofrece al momento de configurar la capa gratuita de servicios. Fuente: <https://aws.amazon.com/es/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc>

Hipervisores

Los hipervisores son *software* que se encargan de separar los recursos físicos de los entornos virtuales. Estos se pueden instalar en un *hardware* concreto como servidor o hacer parte de un sistema operativo existente. En el primer caso, es el mecanismo que utilizan las grandes empresas dedicadas a la virtualización. En el segundo caso, se refiere a lo que se suele hacer a la hora de virtualizar algún sistema operativo sobre un ordenador con sistema operativo funcionando.

El hipervisor gestiona los recursos físicos y los divide de tal forma que todos los entornos virtuales puedan usarlos. Estos recursos se comparten según las demandas de cada virtualización y del propio entorno físico donde están alojadas. Cuando un usuario interactúa con una virtualización, lo hace dentro del entorno virtual conocido comúnmente como máquina de *guest* o máquina virtual.

La máquina virtual funciona como un fichero de datos. El hecho de que sea un fichero sin más dentro del sistema de archivos permite que la máquina virtual se pueda llevar

de un ordenador a otro. Una vez copiado en otro ordenador, al abrirla esta se adapta al nuevo *host* y usa sus recursos para poder funcionar. En esta portabilidad radica la importancia de los entornos virtualizados, ya que permiten que un sistema virtualizado creado en un *host* pueda ser llevado a otro *host* y ejecutarse sin problema alguno.

Tipología de la virtualización

La virtualización se divide en cuatro categorías, que son:

- ▶ **Virtualización de escritorio:** permite que un servidor centralizado ofrezca y administre escritorios de forma individual. Es común confundirla con la virtualización de los sistemas operativos. Sin embargo, en la virtualización de los escritorios existe la figura de administrador o herramienta de administración automatizada que configura los entornos simulados de escritorio en cientos de máquinas físicas al mismo tiempo. En un entorno de escritorio tradicional es necesario instalar herramientas, configurar y actualizar cada máquina una a una; mientras que con la virtualización de escritorios los administradores desde un único punto llevan a cabo estas tareas de forma masiva en todos los escritorios virtuales.
- ▶ **Virtualización de red:** se caracteriza porque divide el ancho de banda de una red en canales independientes que se asignan a servidores o dispositivos específicos. La virtualización de las funciones de red (NFV) permite separar las funciones de una red para luego distribuirlas entre los entornos correspondientes. Esta virtualización reduce la cantidad de componentes físicos tales como commutadores, enruteadores, servidores, cables, centrales, entre otros, necesarios para crear redes independientes.
- ▶ **Virtualización de software:** se encarga de separar las aplicaciones del *hardware* y el sistema operativo. En el caso de la virtualización del sistema operativo, esta se lleva a cabo en el *kernel* (los administradores de tareas de los sistemas operativos).

De esta manera, es posible ejecutar entorno Linux y Windows de forma paralela en una misma máquina. Por otra parte, la virtualización de un servidor permite ejecutar más funciones específicas e implica dividirlo, para que los elementos se puedan utilizar para realizar varias funciones a la vez.

- ▶ **Virtualización de datos o almacenamiento:** combina recursos de almacenamiento en red y los muestra como un solo dispositivo de almacenamiento accesible para varios usuarios. La virtualización de datos permite a las empresas aplicar sobre ellos capacidad de procesamiento, reuniéndolos desde varias fuentes y facilitando la integración de nuevos orígenes de datos. Las herramientas de virtualización de los datos trabajan sobre varias fuentes de datos y los tratan como si fuesen un único origen.

10.6. Docker

Docker es un motor de código abierto cuya función principal es facilitar la implementación de aplicaciones en contenedores. Su creador es *Docker Inc*, conocidos en fechas pasadas como *dotCloud Inc*, pioneros de los servicios del *cloud* Plataforma como servicio (PaaS). Está disponible bajo la licencia Apache 2.0.

Docker propone un sistema de implementación de aplicaciones sobre un entorno de ejecución de contenedores virtualizado. Su entorno se caracteriza por ser rápido, ligero y por facilitar la ejecución de su código. Dentro de su método de trabajo, define un flujo eficiente que permite llevar desarrollos de ordenadores personales a entornos de prueba, integración y producción (Schenker, 2018; Turnbull, 2019).

Docker se diferencia de una máquina virtual por su sencillez y requerimientos mínimos para ejecutar un *host*. De hecho, es posible iniciar Docker con un *host* mínimo que solo ejecute un *kernel* de Linux compatible con Docker (Schenker, 2018).

Actualmente, existen muchas distribuciones de *kernel* Linux totalmente compatibles con Docker.

Cómo funciona Docker

En un principio Docker usaba **LinuX Containers** (LXC) para crear sus contenedores, pero posteriormente cambió a **runC** (libcontainer). Este último se ejecuta en el mismo sistema operativo que su *host*, permitiendo compartir los recursos del sistema operativo del *host*. El sistema de archivos que utiliza se basa en capas (AuFS) y permite gestionar las redes del mismo.

AuFS es un sistema de archivos basado en capas, lo cual permite que se fusionen capas de solo lectura y capas de escritura. En una capa de solo lectura se pueden almacenar las partes comunes del sistema operativo y luego dotar a cada contenedor con su propio montaje para escritura. Esto a su vez permitiría compartir entre todos sus contenedores la capa del sistema operativo.

Supón que dispones de una imagen de contenedor de 2 GB. Si se usa una máquina virtual completa, se necesitan los 2 GB multiplicados por la cantidad de máquinas virtuales que demandes. Al usar Docker y AuFS, puedes compartir la mayor parte de esos 2 GB entre todos los contenedores y, entre 30 contenedores, no llegar a ocupar más de 2 GB de espacio en el sistema operativo de los contenedores (suponiendo, claro, que todos ejecutan el mismo sistema operativo).

Máquinas virtuales vs. Docker

Una máquina virtual completa utiliza su propio conjunto de recursos asignados y comparte los mínimos recursos posibles. Está mucho más aislada y por ello es muy pesada (requiere más recursos propios). Docker ofrece menos aislamiento, pero sus contenedores son más livianos (requieren menos recursos). Esto último permite que

se ejecuten muchos contenedores en un mismo *host* sin destrozarlo (McKendrick et al., 2017).

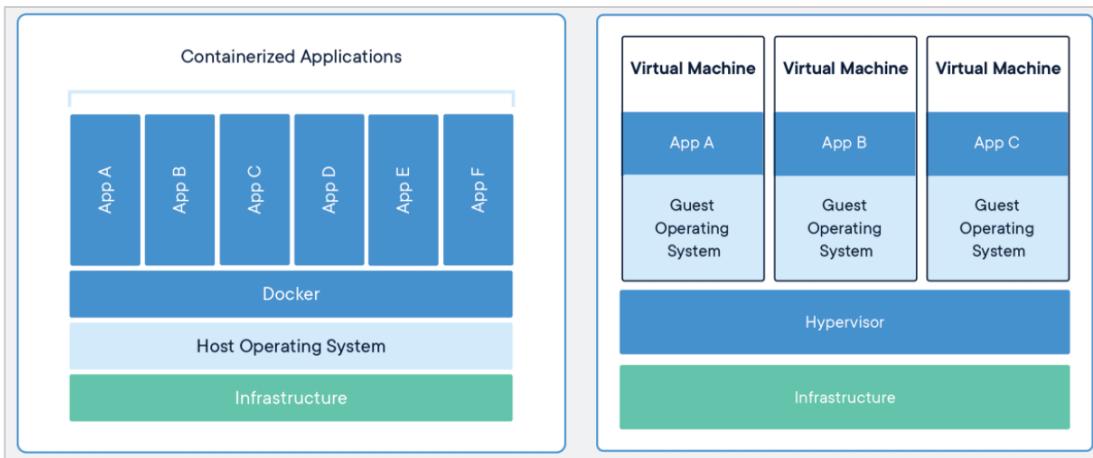


Figura 7. Implementación de un contenedor Docker vs. una máquina virtual. Fuente: https://www.docker.com/resources/what-container#/package_software

En el caso de máquinas virtuales completas, en un host es probable que el número de máquinas ejecutándose a la vez sea muy limitado. Por otro lado, una máquina virtual suele tardar unos minutos en iniciarse (como cualquier sistema operativo común), mientras que los contenedores Docker/LXC/runC tardan unos pocos segundos.

Existen pros y contras para cada tipo de virtualización. Si se necesita un aislamiento total con recursos garantizados, una máquina virtual completa es la mejor opción. Si por el contrario solo desea aislar procesos concretos entre sí y desea ejecutar muchos de ellos en un único host de tamaño razonable, Docker/LXC/runC podría ser de mucha utilidad.

Componentes de Docker

Los componentes centrales que constituyen Docker son:

- ▶ El cliente y el servidor de Docker.
- ▶ Imágenes.

- ▶ Registros.
- ▶ Contenedores.

Cliente y servidor de Docker

Docker es una aplicación cliente-servidor. El cliente se comunica con el servidor o demonio de Docker, que es quien hace todo el trabajo. Docker incluye cliente mediante línea de comandos, el cual implementa una API RESTful completa. Tanto el servidor (demonio) como el cliente de Docker se pueden ejecutar en un mismo *host*. También es posible que el cliente Docker local se conecte a un demonio remoto alojado en otro *host* (Turnbull, 2019).

Imágenes

Las imágenes son los componentes básicos de Docker. Los contenedores se ejecutan a partir de imágenes concretas. Las imágenes definen el ciclo de vida de Docker y se basan en un formato en capas. Dichas capas utilizan sistemas de archivos que se crean paso a paso mediante una serie de instrucciones.

Las imágenes son el «código fuente» de su contenedor, son portables y, lo más importante, se pueden compartir, almacenar y actualizar.

Registros

Existen dos tipos de registro: públicos y privados. Docker almacena las imágenes que crea en registros. Docker, Inc., utiliza un registro público de imágenes, llamado Docker Hub. Es recomendable crear una cuenta en Docker Hub y usarla para compartir y almacenar tus propias imágenes.

Considera Docker Hub como un repositorio de imágenes al que nos conectamos para descargar dichas imágenes. En el último recuento, existen más de 10 000 imágenes que otros usuarios han creado y compartido.

También es posible almacenar imágenes privadas para compartirlas solo con quien consideres oportuno. Estas imágenes pueden incluir código fuente u otra información que deseas mantener segura o solo compartir con otros miembros de tu equipo u organización.

Contenedores

Docker permite crear e implementar contenedores donde luego empaqueta aplicaciones y servicios. A partir de las imágenes, los contenedores se lanzan con uno más proceso en ejecución. Algunos autores como James Turnbull (2019) definen un contenedor como:

- ▶ Un formato de imagen.
- ▶ Un conjunto de operaciones estándar.
- ▶ Un entorno de ejecución.

El concepto de contenedor es similar al de contenedor de mercancía, utilizado para el transporte a nivel internacional. La diferencia principal es que Docker mueve aplicaciones *software* en sus contenedores. Cada contenedor contiene una imagen de *software* sobre la que es posible realizar un conjunto de operaciones, por ejemplo, crear, iniciar, detener, reiniciar y destruir (Turnbull, 2019).

Docker no tiene en cuenta qué contiene el contenedor. Al manipularlo, independientemente de si un contenedor es una aplicación web, una base de datos u otra cosa, cada contenedor se carga igual que cualquier otro. Dicho contenedor se cargará en su ordenador personal, en un servidor o en un registro, para luego descargarlo en un servidor físico o virtual, o implementarlo en un clúster de *hosts* Amazon EC2.

10.7. Manos a la obra con Docker

En este apartado llevarás a cabo la instalación de Docker sobre Windows. Posteriormente, crearás un contenedor de desarrollo con MongoDB en su versión más reciente.

Instalar Docker Desktop en Windows

- ▶ En Google, ubica el ejecutable de Docker Desktop para Windows. Cuando lo hayas descargado, ejecuta dicho instalador. Visita la página de Docker Hub para más información.



Figura 8. Opciones de instalación disponibles en Docker Hub.

Si aún no has descargado el instalador (Docker Desktop Installer.exe), puedes obtenerlo de Docker Hub. Por lo general, se descarga en la carpeta de Descargas, aunque también puedes ejecutarlo desde la barra de descargas recientes en la parte inferior de tu navegador web.

- ▶ La instalación en algún momento te pedirá que habilites la opción **Enable Hyper-V Windows Features**. Asegúrate que esté seleccionada en la página Configuración.
- ▶ Sigue las instrucciones del instalador, autoriza las diferentes opciones de acceso a recursos de tu ordenador.
- ▶ Si tu cuenta de **administrador** es diferente a tu cuenta de usuario, debes agregar el usuario al grupo **docker-users**. Ejecuta Computer Management como administrador y dirígete a Usuarios y Grupos locales> Grupos> docker-users. Haz

clic derecho para agregar el usuario al grupo. Cierra la sesión y vuelve a iniciarla para que los cambios surtan efecto.

Iniciar Docker Desktop

Docker Desktop no se inicia automáticamente después de la instalación. Para iniciar Docker Desktop, debes buscar Docker en Inicio y seleccionar Docker Desktop en los resultados de la búsqueda.

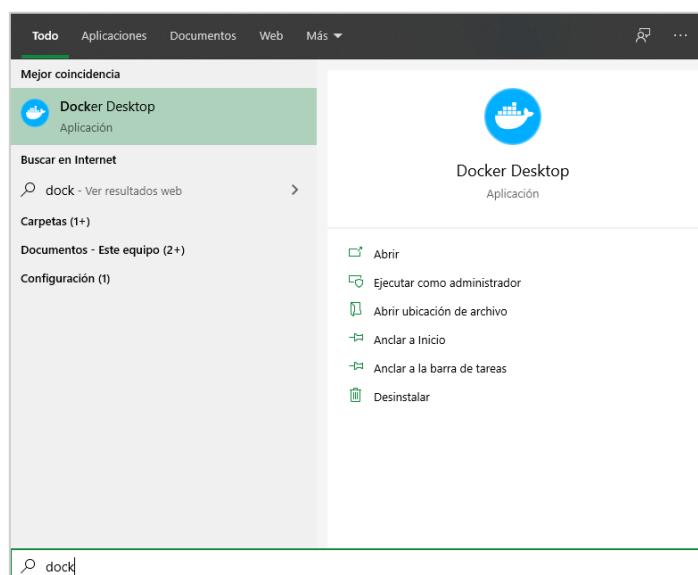


Figura 9. Menú Inicio y la opción Docker Desktop después de instalado.

Cuando el ícono de la ballena en la barra de tareas permanezca fijo, Docker Desktop estará en funcionamiento y listo para ser usado desde cualquier consola.

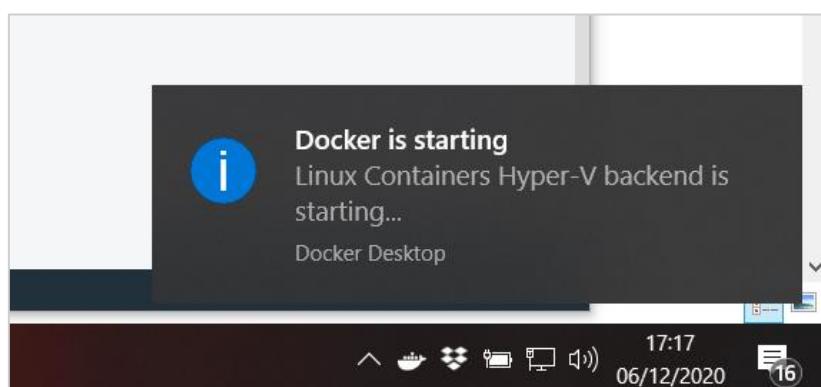


Figura 10. Aviso de Docker en ejecución sobre Windows.

Cuando se completa la inicialización, Docker Desktop inicia el tutorial de incorporación. El tutorial incluye un ejercicio simple para crear una imagen de Docker de ejemplo, ejecutarla como un contenedor, empujar y guardar la imagen en Docker Hub.

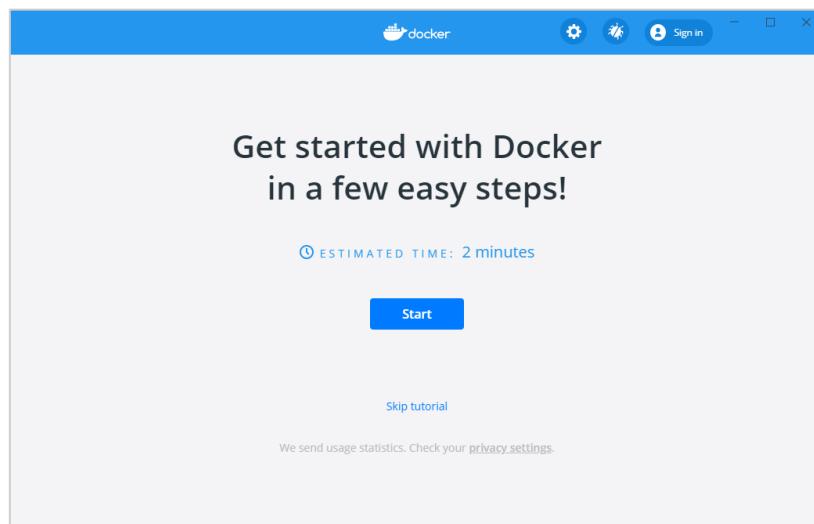


Figura 11. Inicio de Docker Desktop sobre Windows.

Las instrucciones de instalación se basan en las indicaciones de la página oficial de Docker Inc. Consulta la URL oficial para mayor información: [Docker Inc Instalación](#).

10.8. Iniciar un contenedor con MongoDB

- ▶ **Paso 1:** instala la imagen de MongoDB en su Servidor Docker Desktop.

```
> docker pull mongo
```

```
C:\Users\mfcardenas>docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
f22ccc0b8772: Pull complete
3cf8fb62ba5f: Pull complete
e80c964ecea6: Pull complete
329e632c35b3: Pull complete
3e1bd1325a3d: Pull complete
4aa6e3d64a4a: Pull complete
035bca87b778: Pull complete
874e4e43cb00: Pull complete
08cb97662b80: Pull complete
f623ce2ba1e1: Extracting [=====]>
] 80.77MB/142.7MB
f100ac278196: Download complete
461b064aece5: Download complete
```

Figura 12. Instalación de imagen de MongoDB en Docker.

- **Paso 2:** si deseas instalar una versión concreta, debes indicar la versión correspondiente, por ejemplo:

```
> docker pull mongo:4.2.2
```

- **Paso 3:** consulta todas las imágenes que tienes instaladas en tu ordenador:

```
> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mongo	latest	609301053242	40 hours ago	493MB
cassandra	latest	8baadf8d390f	10 days ago	405MB
redislabs/redis	latest	69e5ecb7e5a1	5 weeks ago	1.2GB
dremio/dremio-oss	latest	5fcfd928fed5a	2 months ago	1.17GB
docker/getting-started	latest	be9b467ffb6c	5 months ago	26.5MB

Figura 13. Consulta de imágenes disponibles Docker sobre Windows.

- **Paso 4:** crea un directorio donde almacenar los documentos de la base de datos.

Por ejemplo: E:/MongoDB/mongodata.

- **Paso 5:** despliega una instancia de MongoDB como contenedor; para ello escribe la siguiente instrucción:

```
> docker run -it -v e:/MongoDB/mongodata:/data/db --name mongodb -d mongo
```

- **-it:** proporciona una *shell* para comunicarse con el contenedor.
- **-v:** asocia la ruta local **E:/MongoDB/mongodata** con la ruta del contenedor **/data/db**. **Cuidado:** Docker te pedirá permisos para acceder al disco E.
- **-d:** el contenedor se inicia en *background* (podemos hacer otras cosas en la terminal).
- **--name:** nombre del contenedor.

```
C:\Users\mfcardenas>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
C:\Users\mfcardenas>docker run -it -v e:/MongoDB/mongodata:/data/db --name mongodb -d mongo
72b9d3fdc0d90783c97a809df3ca6bb287ce2e75702bccafa232e958fd2b99f1
C:\Users\mfcardenas>
```

Figura 14. Visualizar imágenes en Docker sobre Windows.

► **Paso 6:** consulta los contenedores activos en tu servidor Docker:

```
> docker ps
```

```
C:\Users\mfcardenas>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
72b9d3fdc0d9        mongo              "docker-entrypoint.s..."   About a minute ago   Up About a minute   27017/tcp
NAMES
C:\Users\mfcardenas>
```

Figura 15. Consultar imágenes de Docker sobre Windows.

► **Paso 7:** realiza las siguientes operaciones con tu nuevo contenedor.

- Para el contenedor.
- Vuelve a ponerlo en marcha.
- Vuelve a parar el contenedor.
- Borra dicho contenedor.

```
> docker stop 72b9d3fdc0d9
> docker start 72b9d3fdc0d9
> docker stop 72b9d3fdc0d9
> docker rm 72b9d3fdc0d9
```

Para manipular el contenedor, se recomienda usar el id del mismo. Este id se asigna al momento de crearse y es la forma como podemos referirnos a él.

```
C:\Users\mfcardenas>docker stop 72b9d3fdc0d9  
72b9d3fdc0d9  
C:\Users\mfcardenas>docker start 72b9d3fdc0d9  
72b9d3fdc0d9  
C:\Users\mfcardenas>docker stop 72b9d3fdc0d9  
72b9d3fdc0d9  
C:\Users\mfcardenas>docker rm 72b9d3fdc0d9  
72b9d3fdc0d9  
C:\Users\mfcardenas>docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES

```
C:\Users\mfcardenas>
```

Figura 16. Detener y borrar un contenedor en Docker sobre Windows.

- **Paso 8:** vuelve a crear el contenedor con la siguiente instrucción:

```
> docker run -it -v e:/MongoDB/mongodata:/data/db -p 27017:27017 --name  
mongodb -d mongo
```

```
C:\Users\mfcardenas>docker run -it -v e:/MongoDB/mongodata:/data/db -p 27017:27017 --name mongodb -d mongo  
fc6d15bf3749dfced2eb848eafedb3a051c39a03abb92e6b0d3399912aa2ff10
```

Figura 17. Ejecutar contenedor de Docker sobre Windows.

Ahora hemos añadido la opción **-p**, la cual permite establecer el puerto de *mongo* en nuestro ordenador y luego el puerto de *mongo* en el contenedor.

- **Paso 9: atención**, es muy probable que, al ejecutar este paso, el contenedor de *mongodb* no se haya levantado correctamente. Comprueba que el contenedor está funcionando (Paso 6). Si la columna STATUS indica «Exit» en su contenido o el puerto PORTS está vacío, significa que tu contenedor no está levantado.

- **Paso 10:** comprueba que el contenedor se creó, pero no se levantó:

```
> docker ps -al
```

```

C:\Users\mfcardenas>docker run -it -v e:/MongoDB/mongodata:/data/db -p 27017:27017 --name mongodb -d mongo
C:\Users\mfcardenas>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
C:\Users\mfcardenas>docker ps -al
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
e784c928b030        mongo              "docker-entrypoint.s..."   13 seconds ago    Exited (14)   11 seconds ago   mongodb
C:\Users\mfcardenas>

```

Figura 18. Ejecutar contenedor de Docker sobre Windows especificando puertos.

- ▶ **Paso 11:** consulta el *log* del contenedor creado, utiliza la siguiente instrucción:

```
> docker logs mongodb
```

```

C:\Users\mfcardenas>docker logs mongodb
{"t": {"$date": "2020-12-06T17:14:24.217+00:00"}, "s": "I", "c": "CONTROL", "id": 23285, "ctx": "main", "msg": "Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}, {"t": {"$date": "2020-12-06T17:14:24.219+00:00"}, "s": "W", "c": "ASIO", "id": 22601, "ctx": "main", "msg": "No TransportLayer configured during NetworkInterface startup"}, {"t": {"$date": "2020-12-06T17:14:24.219+00:00"}, "s": "I", "c": "NETWORK", "id": 4648601, "ctx": "main", "msg": "Implicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient and tcpFastOpenQueueSize."}, {"t": {"$date": "2020-12-06T17:14:24.228+00:00"}, "s": "I", "c": "STORAGE", "id": 4615611, "ctx": "initandlisten", "msg": "MongoDB starting", "attr": {"pid": 1, "port": 27017, "dbpath": "/data/db", "architecture": "64-bit", "host": "e784c928b030"}}, {"t": {"$date": "2020-12-06T17:14:24.228+00:00"}, "s": "I", "c": "CONTROL", "id": 23403, "ctx": "initandlisten", "msg": "Build Info", "attr": {"buildInfo": {"version": "4.4.2", "gitVersion": "b673dc538b6...23d06005f84279b6e", "openSSLVersion": "OpenSSL 1.1.1 I Sep 2018", "modules": {}, "allocator": "tcmalloc", "environment": {"distmod": "ubuntu1804", "distarch": "x86_64", "osArch": "x86_64"}}, {"t": {"$date": "2020-12-06T17:14:24.228+00:00"}, "s": "I", "c": "CONTROL", "id": 51765, "ctx": "initandlisten", "msg": "Operating System", "attr": {"os": {"name": "Ubuntu", "version": "18.04"}}, {"t": {"$date": "2020-12-06T17:14:24.228+00:00"}, "s": "I", "c": "CONTROL", "id": 21051, "ctx": "initandlisten", "msg": "Options set by command line", "attr": {"options": {"net": {"bindIp": "0.0.0.0"}}, {"t": {"$date": "2020-12-06T17:14:24.228+00:00"}, "s": "I", "c": "STORAGE", "id": 22270, "ctx": "initandlisten", "msg": "Storage engine to use detected by data files", "attr": {"dbpath": "/data/db", "storageEngine": "wiredTiger"}}, {"t": {"$date": "2020-12-06T17:14:24.228+00:00"}, "s": "I", "c": "STORAGE", "id": 22235, "ctx": "initandlisten", "msg": "Opening WiredTiger", "attr": {"config": "create,cache_size=4471M,session_max=33000,eviction=(threads min=4,threads max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close idle time=100000,close_scan_interval=10,close_handle_minimum=250),statistics_log=(wait=0),verbose=[recovery_progress,checkpoint_progress,compact_progress]"}, {"t": {"$date": "2020-12-06T17:14:24.665+00:00"}, "s": "E", "c": "STORAGE", "id": 22435, "ctx": "initandlisten", "msg": "WiredTiger error", "attr": {"error": 1, "message": "[1607274864:665945][1:0x7f5fab100ac0], fileWiredTiger.wt, connection: [0:0x7f5fbab00ac0], fileWiredTiger.wt, handle: open: Operation not permitted"}}, {"t": {"$date": "2020-12-06T17:14:24.721+00:00"}, "s": "E", "c": "STORAGE", "id": 22435, "ctx": "initandlisten", "msg": "WiredTiger error", "attr": {"error": 1, "message": "[1607274864:6659518][1:0x7f5fab100ac0], fileWiredTiger.wt, connection: [0:0x7f5fbab00ac0], fileWiredTiger.wt, handle: open: Operation not permitted"}}, {"t": {"$date": "2020-12-06T17:14:24.721+00:00"}, "s": "E", "c": "STORAGE", "id": 22435, "ctx": "initandlisten", "msg": "WiredTiger error", "attr": {"error": 1, "message": "[1607274864:721880][1:0x7f5fab100ac0], fileWiredTiger.wt, connection: [0:0x7f5fbab00ac0], fileWiredTiger.wt, handle: open: Operation not permitted"}}, {"t": {"$date": "2020-12-06T17:14:24.723+00:00"}, "s": "E", "c": "STORAGE", "id": 22347, "ctx": "initandlisten", "msg": "Failed to start up WiredTiger under any compatibility version. This may be due to an unsupported upgrade or downgrade."}, {"t": {"$date": "2020-12-06T17:14:24.723+00:00"}, "s": "F", "c": "STORAGE", "id": 28595, "ctx": "initandlisten", "msg": "Terminating.", "attr": {"reason": 1: Operation not permitted"}}, {"t": {"$date": "2020-12-06T17:14:24.723+00:00"}, "s": "F", "c": "STORAGE", "id": 23091, "ctx": "initandlisten", "msg": "Fatal assertion", "attr": {"msgId": 28595, "file": "src/mongo/db/storage/wiredtiger/wiredtiger_kv_engine.cpp", "line": 1123}], {"t": {"$date": "2020-12-06T17:14:24.723+00:00"}, "s": "F", "c": "F", "id": 23092, "ctx": "initandlisten", "msg": "\n\n***aborted after fassert() failure\n\n"}

```

Figura 19. Visualizar el *log* del contenedor de Docker sobre Windows.

- ▶ **Paso 12:** ¿puedes deducir cuál es el problema? Tómate tu tiempo y repasa mentalmente los pasos 4 a 11. Revisa la información del *log* de MongoDB y deduce cuál puede ser el problema. No continúes con el siguiente paso hasta que no identifiques el problema tú mismo.

- ▶ **Paso 13:** ¡enhorabuena! Has dado con el error. Asignaste la ruta E:/MongoDB/mongodata, como directorio para los datos. Al hacerlo, MongoDB creó allí copia de todos tus datos. Cuando borraste el contenedor, no borraste lo que había en este directorio y, al crear un nuevo contenedor, MongoDB se encontró allí con el directorio ocupado y no pudo crear los nuevos ficheros de la base de datos. Borra el contenedor creado (revisa el Paso 7 y el Paso 10), limpia este directorio o asigna otro diferente y vuelve a crear el contenedor (Paso 8).

```

C:\Users\mfcardenas>docker ps -al
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
e784c928b030        mongo              "docker-entrypoint.s..."   11 minutes ago    Exited (14) 11 minutes ago      mongodb

C:\Users\mfcardenas>docker rm e784c928b030
e784c928b030

C:\Users\mfcardenas>docker run -it -v e:/MongoDB/mongodata:/data/db -p 27017:27017 --name mongodb -d mongo
9723c54e59ded31793825247260547026504dbb9bb26cbe1fd4b86c46561cde7

C:\Users\mfcardenas>docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
9723c54e59de        mongo              "docker-entrypoint.s..."   3 seconds ago     Up 3 seconds          0.0.0.0:27017->27017/tcp   mongodb

C:\Users\mfcardenas>

```

Figura 20. Visualizar imágenes ocultas o con errores en Docker sobre Windows.

► **Paso 14:** conéctate a la consola de *mongodb* (tu contenedor) y ejecuta algunas instrucciones como las siguientes.

- Lista las bases de datos existentes.
- Crea una base de datos llamada *productos*.
- Inserta en esta base de datos el documento `{"nombre": "teclado", tipo: "hardware", coste: 125.12, moneda: "EURO"}`). Utiliza exit dos veces para salir de *mongo* y la consola Bash.

> docker logs mongodb

Escribe mongo en la terminal Bash.

```

C:\Users\mfcardenas>docker exec -it mongodb bash
root@9723c54e59de:/# mongo
MongoDB shell version v4.4.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("154badce-e91a-4fa4-8cef-9dbdfd2d238a") }
MongoDB server version: 4.4.2
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
  https://community.mongodb.com
---
The server generated these startup warnings when booting:
  2020-12-06T17:26:37.035+00:00: Access control is not enabled for the database. Read and write ac
dicted
---
---
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display
  metrics about your deployment (disk utilization, CPU, operation statistics, etc).

  The monitoring data will be available on a MongoDB website with a unique URL accessible to you
  and anyone you share the URL with. MongoDB may use this information to make product
  improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>

```

Figura 21. Ejecutar contenedor MongoDB de Docker sobre Windows.

```

--> use productos
switched to db productos
> db.productos.insert({ "nombre": "teclado", tipo: "hardware", coste: 125.12, moneda: "EURO"})
WriteResult({ "nInserted": 1 })
> db.productos.find()
{ "_id" : ObjectId("5fcfd16157ff1e6dabe97020c"), "nombre" : "teclado", "tipo" : "hardware", "coste" : 125.12, "moneda" : "EURO" }
>

```

Figura 22. Usar el contenedor MongoDB de Docker sobre Windows.

► **Paso 15:** utiliza MongoDB Compass para conectarte a tu contenedor. MongoDB Compass se instala por defecto con el instalador de MongoDB. En los temas anteriores es probable que ya hayas hecho la instalación de MongoDB y, por ende, de Compass.

- Si esto es así, para el servicio local de MongoDB en tu ordenador (Servicios en Windows, buscar MongoDB en la lista y Detener).
- Luego, ejecuta MongoDB Compass y New Connections, da clic en Connect.
- Repara en que no indicas la URL de conexión, MongoDB usa la URL de conexión por defecto. La URL de conexión del contenedor coincide con tu URL de conexión al servidor de *mongo* que instalaste en los temas anteriores. Esto es así porque nunca cambiaste el puerto por defecto.
- Desde Compass, consulta la base de datos *productos* y la colección que creaste en el Paso 14. Si no has sido capaz de llegar a este punto, consulta en el Foro todas tus dudas.

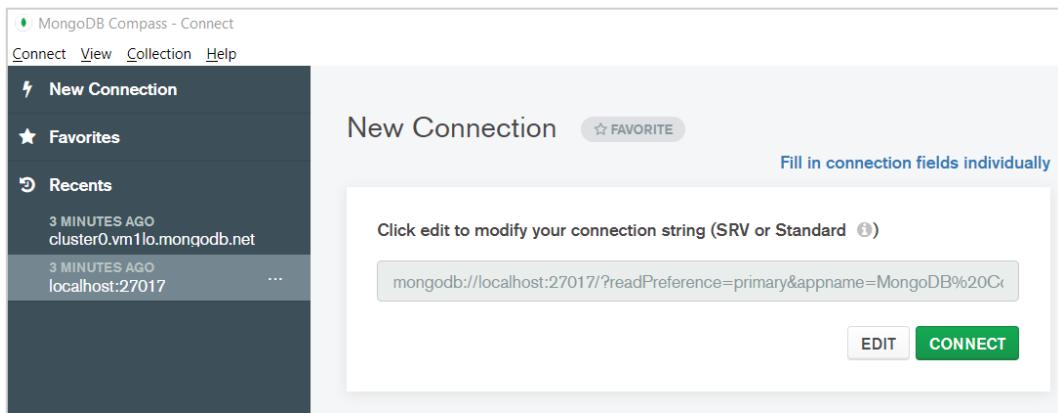


Figura 23. Conectar MongoDB Compass local con una instancia de Docker sobre Windows.

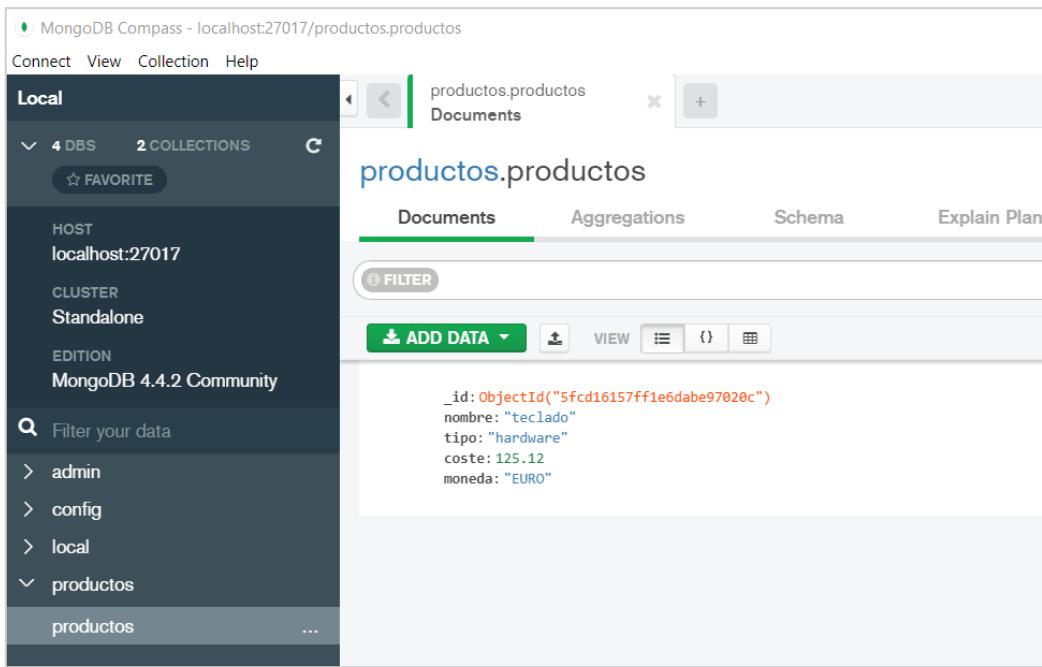


Figura 24. Visualizar datos de MongoDB en instancia de Docker sobre Windows.

27017 es el puerto predeterminado para instancias *mongod* y *mongos*.

27018 es el puerto predeterminado para *mongod* cuando se ejecuta con la opción de línea de comandos `--shardsvr` o el valor de `shardsvr` para la configuración `clusterRole` en un archivo de configuración.

27019 es el puerto predeterminado para *mongod* cuando se ejecuta con la opción de línea de comandos `--configsvr` o el valor `configsvr` para la configuración `clusterRole` en un archivo de configuración.

Después de leer todo el tema, en el vídeo «*Cloud computing: aspectos claves*» podrás repasar la definición de *cloud*, los modelos de *cloud* que existen y los tipos de servicios que se ofrecen a través de él.



Vídeo 1. *Cloud computing*: aspectos claves.

Accede al vídeo a través del aula virtual

10.9. Referencias bibliográficas

McKendrick, R., Raj, P., Chelladhurai, J. S., y Singh, V. (2017). *Docker Bootcamp*. Packt Publishing.

Oracle. (2021). ¿Qué significa «Cloud Computing»? [Página web].

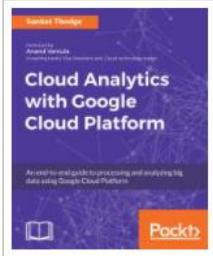
<https://www.oracle.com/es/cloud/what-is-cloud-computing/#:~:text=En%20t%C3%A9rminos%20sencillos%2C%20Cloud%20computing,pagar%20por%20lo%20que%20consumen.>

Turnbull, J. (2019). *The Docker book: containerization is the new virtualization*. James Turnbull.

A fondo

Cloud computing: Cloud Analytics with Google Cloud Platform

Thodge, S. (2018). *Cloud Analytics with Google Cloud Platform: An end-to-end guide to processing and analyzing big data using Google Cloud Platform*. Packt Publishing.

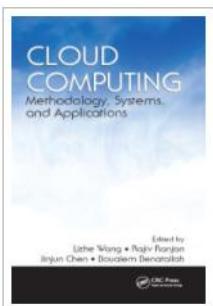


Este libro combina el poder de la analítica y la computación en la nube para obtener información más rápida y eficiente. Describe el concepto de analítica en la nube y cómo las organizaciones lo utilizan; también muestra las consideraciones de diseño mientras aplica una solución de analítica en la nube.

Accede al documento a través de la Biblioteca Virtual UNIR

Cloud computing: methodology, systems, and applications

Wang, L., Ranjan, R., Chen, J. y Benatallah, B. (2011). *Cloud computing: methodology, systems, and applications*. Taylor & Francis Group.



En este libro se cuenta cómo la computación en la nube ha creado un cambio del uso de *hardware* físico y plataformas habilitadas por *software* administradas localmente al de servicios virtualizados alojados en la nube. La nube ensambla grandes redes de servicios virtuales, que incluyen *hardware* (CPU, almacenamiento y red) y recursos de *software* (bases de datos, sistemas de cola de mensajes, sistemas de monitoreo y平衡adores de carga).

Accede al documento a través de la Biblioteca Virtual UNIR

Post de ayuda de Docker Inc.

Docker Inc. (2020). Home [Página web]. <https://docs.docker.com/>

Mortensen, P. (28 de agosto de 2018). How is Docker different from a virtual machine? *Stack overflow* [Mensaje en un foro].

<https://stackoverflow.com/questions/16047306/how-is-docker-different-from-a-virtual-machine>

Documentación de apoyo donde se describen las características de los contenedores en Docker.



Documentación oficial: Docker Hub

Docker Inc. (2020). Docker Hub Quickstart [Página web].

<https://docs.docker.com/docker-hub/>

Docker Hub es un servicio proporcionado por Docker para buscar y compartir imágenes de contenedores con tu equipo. Es el repositorio de imágenes de contenedores más grande del mundo con una variedad de fuentes de contenido que incluyen desarrolladores de comunidades de contenedores, proyectos de código abierto y proveedores de *software* independientes (ISV) que crean y distribuyen su código en contenedores.

Curso de introducción a Docker

Fazt Code. (10 de mayo de 2020). *Docker, Curso Práctico para principiantes (desde Linux)* [Archivo de vídeo]. <https://www.youtube.com/watch?v=NVvZNmfqg6M>



Este vídeo es una introducción práctica a Docker desde Linux. En este ejemplo se aprende a instalar este programa para la creación y administración de contenedores. También se enseñan los comandos básicos.

Bibliografía

Kane, S. P., & Matthias, K. (2018). *Docker: up & running: shipping reliable containers in production*. O'Reilly Media.

Kropp, A., & Torre, R. (2020). Docker: containerize your application. En *Computing in Communication Networks* (pp. 231-244). Academic Press.

Miell, I., & Sayers, A. (2019). *Docker in practice*. Simon and Schuster.

Nickoloff, J., & Kuenzli, S. (2019). *Docker in action*. Simon and Schuster.

Surovich, S., & Boorshtein, M. (2020). *Kubernetes and Docker-An Enterprise Guide: Effectively containerize applications, integrate enterprise systems, and scale applications in your enterprise*. Packt Publishing Ltd.

1. ¿Cuáles son los beneficios del *cloud computing*?

- A. Escalabilidad, flexibilidad, seguridad, pago por uso...
- B. Flexibilidad, pago por uso y seguridad.
- C. Escalabilidad y portabilidad.
- D. Ninguno de los anteriores.

Los beneficios del *cloud computing* son muchos, pero los más mencionados son estos.

2. ¿Cuántos modelos de *cloud* existen?

- A. Dos: público y privado.
- B. Tres: público, privado y de pago.
- C. Tres: público, privado e híbrido.
- D. Ninguno de los anteriores.

Los tres modelos de *cloud* son público, privado e híbrido. Los privados son de pago.

3. Una afirmación verdadera sobre SaaS:

- A. SaaS es ofrecer servicios de sistemas operativos virtualizados.
- B. Se centra en alojar e implementar *software* del cliente en el *cloud* para que este luego lo use a demanda.
- C. Significa *Software como Servicio* y ofrece al cliente licencia de todo tipo de *software*.
- D. El cliente usa este servicio como repositorio para guardar todo el *software* que compra la compañía y así no instalarlo en sus servidores principales.

El cliente aloja en *cloud* solo aquellos programas que le representen una alta demanda de recursos en local. Estos recursos pueden ser capacidad de cómputo, recursos físicos, etc.

4. ¿En qué se parecen PaaS e IaaS?

- A. En nada, cada uno ofrece servicios totalmente diferentes.
- B. PaaS ofrece determinados servicios como parte de un contexto de plataforma; IaaS, por su parte, puede que ofrezca servicios similares, pero de forma independiente.
- C. Ambos ofrecen servicios parecidos, solo que uno le da un enfoque más corporativo y el otro se centra más en resolver problemas puntuales del cliente.
- D. Ninguna de las anteriores es correcta.

Tal y como se indica, los servicios podrán ser similares en ciertos casos, pero el enfoque del primero va orientado a que el cliente disfrute de las ventajas de toda una plataforma de trabajo y no de servicios individualizados.

5. ¿Qué clase de servicio se podría considerar Google Docs?

- A. IaaS.
- B. PaaS.
- C. SaaS.
- D. PaaS y SaaS.

Google Docs ofrece *software* a sus clientes mediante el *cloud*, por ende, el modelo de servicio que más se ajusta es SaaS.

6. El encargado de gestionar los recursos físicos y distribuirlos de forma equitativa para que las virtualizaciones hagan uso de ellos es:

- A. Docker.
- B. *Cloud computing*.
- C. SaaS.
- D. Hipervisor.

Esta es la responsabilidad del hipervisor. Docker incorpora un hipervisor, pero existen otros que utilizan otros mecanismos y configuraciones.

7. Los componentes de Docker son:

- A. Los registros.
- B. Las imágenes.
- C. Los contenedores.
- D. Todos los anteriores son correctos.

Los tres anteriores y el cliente-servidor de Docker son sus componentes principales.

8. Para trabajar con contenedores en Docker, ¿qué es lo primero que debemos descargar?

- A. El propio contenedor.
- B. Imágenes.
- C. Registros.
- D. Ninguno de los anteriores.

La imagen posee el *kernel* del sistema operativo sobre el cual se creará el contenedor.

Por lo tanto, es lo primero que debemos descargar.

9. ¿Qué hace el comando `docker rm id_contenedor`?

- A. Borra todos los contenedores que se hayan creado previamente.
- B. Borra el contenedor y limpia la imagen para crear nuevos contenedores.
- C. Borra el contenedor cuyo id corresponde con el descrito siempre que esté detenido.
- D. Borra un contenedor cuyo id es *id_contenedor*.

Docker no permite borrar contenedores que se estén ejecutando, por ello, previamente debemos detener el contenedor y luego borrarlo con esta instrucción.

Es necesario también usar el id del contenedor que se quiere borrar.

10. ¿Qué hace el comando docker images?

- A. Muestra la lista de imágenes creadas o existentes.
- B. Muestra las imágenes con sus correspondientes contenedores asociados.
- C. Muestra las imágenes que se están ejecutando.
- D. Ninguna de las anteriores.

Este comando permite visualizar todas las imágenes creadas hasta el momento en Docker. Con él podrás conocer el repositorio de cada máquina, su etiqueta y el tamaño del espacio ocupado.