



Base de Datos Avanzado II

Curso	Bases de Datos Avanzado II (2398)
Formato	Manual de curso
Autor Institucional	Cibertec
Páginas	120 p.
Elaborador	Cáceres Chávez, Jorge Luis
Revisor de Contenidos	Morales Flores, Gustavo

Índice

Presentación	6
Red de contenidos	7

UNIDAD DE APRENDIZAJE 1: INTRODUCCIÓN A LAS BASES DE DATOS NOSQL

1.1 Tema 1 : Sistemas de gestión de base de datos	10
1.1.1 : Introducción	10
1.1.2 : Modelo de datos relacionales	10
1.1.3 : Modelo de datos no relacionales	11
1.1.4 : Componentes	11
1.2 Tema 2 : NoSQL	13
1.2.1 : ¿Qué es NoSQL?	13
1.2.2 : Teorema CAP	14
1.2.3 : Características	15
1.3 Tema 3 : Tipos de bases de datos NoSQL	17
1.3.1 : Claves / Valores	17
1.3.2 : Documentales	18
1.3.3 : Grafos	19
1.3.4 : Columnas anchas	20
1.3.5 : Ejemplo de bases de datos NoSQL	21
1.3.6 : Ventajas y desventajas	22
1.4 Tema 4 : SQL vs NoSQL	25
1.4.1 : Cargas de trabajo	25
1.4.2 : Modelado de datos	25
1.4.3 : Propiedades ACID	26
1.4.4 : Rendimiento	26
1.4.5 : Escalado	26
1.4.6 : Terminologías	27

UNIDAD DE APRENDIZAJE 2: MONGODB

2.1 Tema 5 : ¿Qué es MongoDB?	30
2.1.1 : Introducción	30
2.1.2 : ¿Por qué utilizar MongoDB?	30
2.1.3 : Ventajas y desventajas	31
2.1.4 : Características	31
2.1.5 : Arquitectura	32
2.1.6 : Estructura	33
2.1 Tema 6 : Entorno de trabajo	34
2.2.1 : Herramientas	34
2.2.2 : Instalación	37
2.2.3 : Configuración	43
2.2.4 : Consola de trabajo	45
2.2.5 : Tipos de datos	49

UNIDAD DE APRENDIZAJE 3: CRUD

3.1 Tema 7 : ¿Qué es CRUD?	52
3.1.1 : CREATE	53
3.1.2 : READ	59
3.1.3 : UPDATE	62
3.1.4 : DELETE	69

UNIDAD DE APRENDIZAJE 4: GESTIÓN DE DATOS

4.1 Tema 8 : Consultas simples	72
4.1.1 : Find ()	72
4.1.2 : FindOne ()	75
4.2 Tema 9 : Consultas avanzadas	76
4.2.1 : Comparación	76
4.2.2 : Existencias	78
4.2.3 : Lógica	80
4.2.4 : Arrays	82
4.2.5 : Subdocumentos	85
4.3 Tema 10 : Curosres	86
4.3.1 : Count ()	86
4.3.2 : Sort ()	86
4.3.3 : Limit ()	87
4.3.4 : Skip ()	87
4.3.5 : toArray ()	88

UNIDAD DE APRENDIZAJE 5: AGGREGATION FRAMEWORK

5.1 Tema 11 : Consultas simples	92
5.1.1 : \$project	92
5.1.2 : \$match	92
5.1.3 : \$group	93
5.1.4 : \$sort	96
5.1.5 : \$limit	97
5.1.6 : \$skip	97
5.2 Tema 12 : Operadores de expresión	98
5.2.1 : Agrupación	98
5.2.2 : Aritméticos	99
5.2.3 : Comparación	99
5.2.4 : Booleanos	100
5.2.5 : Condicionales	101
5.2.6 : Cadenas	102
5.2.7 : Fechas	103

UNIDAD DE APRENDIZAJE 6: OPTIMIZACIÓN DE MONGODB

6.1 Tema 13 : Índices	107
------------------------------	------------

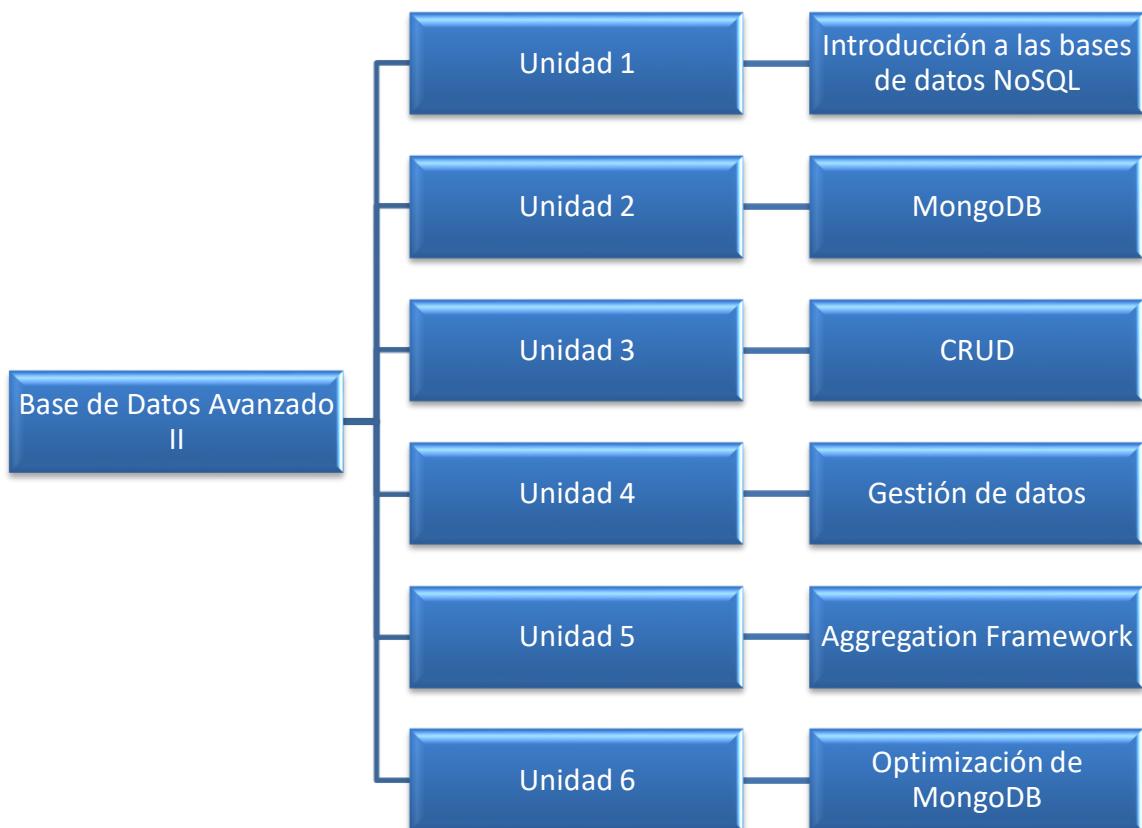
6.1.1 : Introducción	107
6.1.2 : Tipos de índices	108
6.1.3 : Opciones de índices	110
6.1.4 : Utilitarios de base de datos	111
Bibliografía	119

Presentación

Base de Datos Avanzado II es un curso que pertenece a la línea de base de datos y se dicta en las carreras Computación e Informática, y Administración y Sistemas. Brinda un conjunto de herramientas que permite a los alumnos implementar soluciones en una base de datos MongoDB que satisfacen las necesidades de negocio.

El curso es eminentemente práctico consiste en un taller de programación. En primer lugar, se inicia con la instalación y configuración del motor de base de datos no relacionar. Luego, se reconoce las funciones del lenguaje para la gestión de datos. Se concluye con los comandos necesarios mantener la eficiencia en el procesamiento de grandes volúmenes de datos.

Red de contenidos





INTRODUCCIÓN A LAS BASES DE DATOS NOSQL

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno identifica el rol de las bases de datos de tipo NoSQL (no relacionales) en la gestión de los datos masivos.

TEMARIO

1.1 Tema 1 : Sistema de gestión de base de datos

- 1.1.1 : Introducción
- 1.1.2 : Modelo de datos relacionales
- 1.1.3 : Modelo de datos no relacionales
- 1.1.4 : Componentes

1.2 Tema 2 : Bases de datos NoSQL

- 1.2.1 : ¿Qué es NoSQL?
- 1.2.2 : Teorema CAP
- 1.2.3 : Características

1.3 Tema 3 : Tipos de bases de datos NoSQL

- 1.3.1 : Clave / Valor
- 1.3.2 : Documentales
- 1.3.3 : Grafos
- 1.3.4 : Columnas anchas
- 1.3.5 : Ejemplos de bases de datos NoSQL
- 1.3.6 : Ventajas y desventajas

1.4 Tema 4 : SQL vs NoSQL

- 1.4.1 : Cargas de trabajo
- 1.4.2 : Modelado de datos
- 1.4.3 : Propiedades ACID
- 1.4.4 : Rendimiento
- 1.4.5 : Escalado
- 1.4.6 : Terminologías

ACTIVIDADES PROPUESTAS

- Los alumnos realizan las lecturas propuestas.
- Los alumnos identifican las principales diferencias entre los motores de base de datos SQL y NoSQL.
- Los alumnos identifican los motores datos NoSQL que existen en la actualidad.

1.1. SISTEMA DE GESTIÓN DE BASE DE DATOS

1.1.1. Introducción

El objetivo de un sistema de gestión de base de datos (SGBD) consiste en suministrar al usuario las herramientas que le permitan manipular, en términos abstractos, los datos, es decir, no es necesario conocer el modo de almacenamiento físico de los datos en la computadora, ni el método de acceso para administrarlos. Los programas de aplicación operan sobre datos almacenados, manipulando la información y haciendo simple el trabajo de los usuarios.

1.1.2. Modelo de datos relacionales

El modelo de datos relacional plantea la representación la base de datos como una colección de relaciones. Cuando una relación está pensada como tabla de valores, cada fila representa una colección de valores relacionados (Silberschatz, 2002). En este modelo, la fila de la tabla representa un hecho específico que, por lo general, se corresponde con una relación o entidad real. El nombre de la relación y de las columnas se utiliza para ayudar a interpretar el significado de cada uno de los valores de las filas. Y en la actualidad es el enfoque más utilizado, sin embargo, no es eficiente en escenarios de gestión de grandes volúmenes de información.



Figura 1: Modelo de datos relacionales
Fuente.- Elaboración Propia

En la figura 1 la relación se denomina **ALUMNO** porque cada fila representa la información del estudiante específico. Los nombres de columna (Nombre, ID_Alumno, ID_Clase y Especialidad) especifican el modo de interpretar los valores de cada fila en función de la columna en la que se encuentren. Ese esquema de representación es el punto central de la aplicación del modelo relación que un gestor de base de datos relacional implementa al crear una base de datos.

En la terminología formal del modelo de datos relacional que fue estudiado a detalle en cursos previos, una fila recibe el nombre de **tupla**, la cabecera de columna es un **atributo** y el nombre de la tabla **alumno** una relación. El tipo de dato que describe los valores que pueden aparecer en cada columna está representado por un **dominio** de posibles valores tiene una asociación a tablas detalles denominados **tablas maestras** en aplicación de un proceso de normalización.

1.1.3. Modelo de datos no relacionales

No existe un concepto específico de un modelo de datos no relacional, dado que, únicamente, está basado en la necesidad de tener un almacenamiento concreto de un tipo de esquema no estructurado o flexible. Puede parecer una ambigüedad a primera impresión, sin embargo, en los actuales modelos considerados no relacionales, no existe un estándar, ya que introducen o eliminan propiedades de las bases de datos tradicionales a medida que avanza el tiempo.

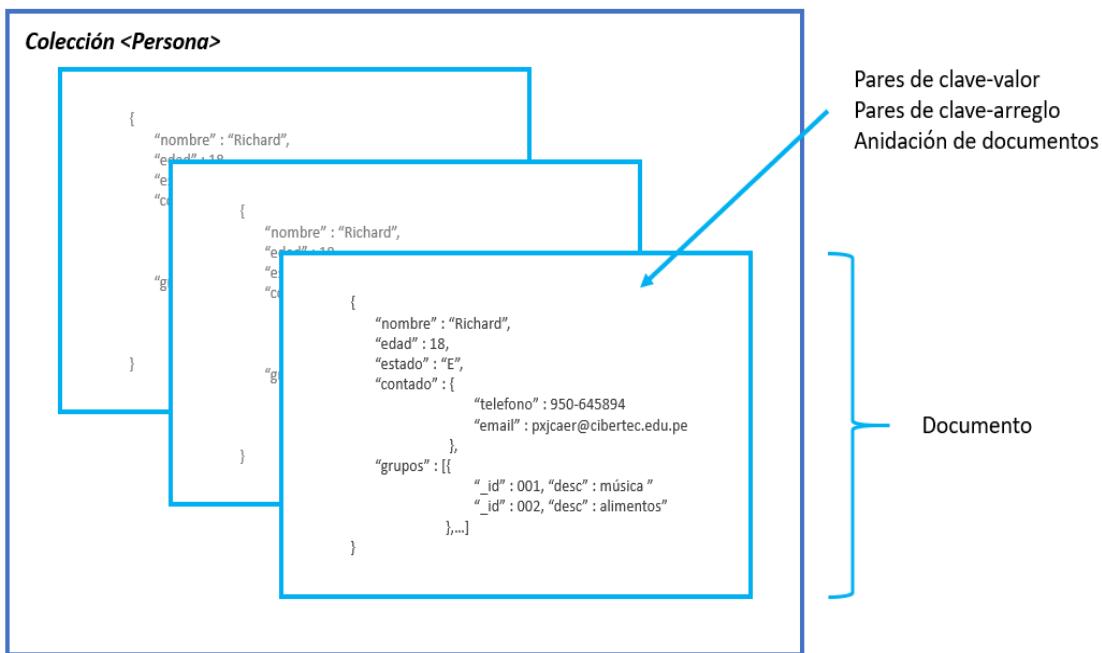


Figura 2: Modelo de datos no relacionales
Fuente.- Elaboración Propia

En la figura 2 la colección se denomina **PERSONA** porque un documento contiene información de un individuo específico. Las claves (Nombre, Edad, Estado, Contacto y Grupo) especifican el modo de interpretar los valores, donde estos últimos pueden ser un **arreglo**, un **documento anidado** o **único valor**.

1.1.4. Componentes

- **El motor de bases de datos** es un elemento que acepta las peticiones de otros subsistemas del SGBD y las convierten en equivalentes físicos (comandos), accediendo a la base de datos y diccionarios en los distintos dispositivos de almacenamientos.
- **El subsistema de definición** de los datos permite crear y mantener el diccionario de datos, así mismo, define la estructura del fichero que soporta a la base de datos.
- **El subsistema de manipulación de datos** ayuda a añadir, cambiar y borrar información de la base de datos.
- **El subsistema de generación de aplicaciones** contiene utilidades para ayudar a los usuarios al desarrollo de aplicaciones personalizadas.

- El **subsistema de administración** ayuda a gestionar la base de datos con funcionalidades de almacenamiento y recuperación, gestión de seguridad, optimización de consultas y control de ocurrencia/cambios. (Homeworkdatabase, 2015).



Figura 3: Base de Datos Relacionales y No Relacionales

Fuente. - Tomado de <https://revistadigital.inesem.es/informatica-y-tics/los-gestores-de-bases-de-datos-mas-usados/>

1.2. BASES DE DATOS NOSQL

1.2.1. ¿Qué es NoSQL?

Las bases de datos NoSQL surgen al ver que las bases de datos relacionales no cumplen con las necesidades actuales de gestión respecto al volumen de datos, gestión de riesgo, agilidad en el desarrollo y velocidad del procesado. En este nuevo escenario, Google, Amazon, Microsoft han experimentado un incremento exponencial del uso de sus soluciones, centrándose los beneficios en la flexibilidad, donde la consistencia deja de ser un esquema fijo para el modelo de datos, la capacidad de almacenamiento una ventaja y no una preocupación central y disminuyendo en los costos operacionales. Esta elasticidad del servicio demuestra una superioridad al momento de almacenar y gestionar grandes volúmenes de información que motiva a distintas empresas a considerar en el mediano plazo la adopción de bases de datos NoSQL.

Se puede decir que la aparición del término NoSQL aparece con la llegada de la web 2.0 ya que hasta ese momento sólo posteaban contenido a la red aquellas empresas que tenían un portal, pero con la llegada de aplicaciones como Facebook, Twitter o YouTube, cualquier usuario podía subir contenido, provocando así un crecimiento exponencial de los datos. Es en este momento cuando empiezan a surgir los primeros problemas en la gestión de la información almacenada en bases de datos relacionales. En un principio, solucionar estos problemas de accesibilidad, las empresas optaron por incrementar el número de servidores, sin embargo, se dieron cuenta que esto no solucionaba el problema. Evidentemente la raíz de modelo relacional escondía una clara desventaja en este nuevo contexto.

Por lo tanto, en este contexto las estructuras relacionales de las bases de datos relacionales o SQL originan problemas debido a la escalabilidad y rendimiento en escenarios concurrentes de miles de usuarios y millones de consultas a información con múltiples formatos (videos, audio, fotos, etc.) que las bases de datos NoSQL solucionan porque sus sistemas de almacenamiento son flexibles y no cumplen con el esquema tradicional **entidad–relación**. Tampoco utilizan una estructura de datos en forma de tablas donde se van almacenando los datos sino hacen uso de otros formatos como clave–valor, mapeo de columnas o grafos.

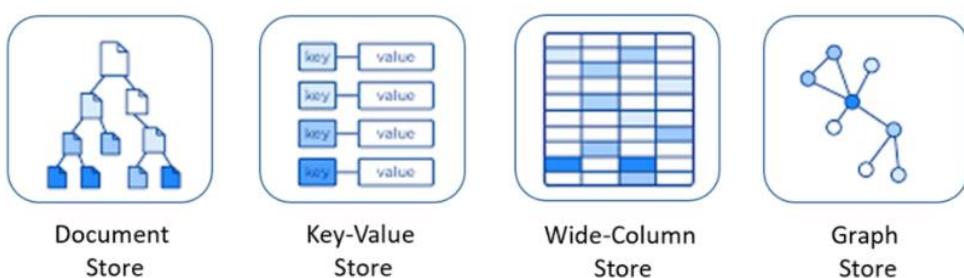


Figura 4: Modelos de datos para bases de datos NoSQL

Fuente. - Tomado de <https://docs.microsoft.com/es-es/dotnet/architecture/cloud-native/relational-vs-nosql-data>

1.2.2. Teorema CAP

Desde que comenzaron a desarrollarse las bases de datos SQL, se convirtieron en un standard en el campo del almacenamiento, sin embargo, el aumento de los volúmenes almacenados y la necesidad de ejecutar consultas en tiempo real sobre grandes cantidades y formas de datos ha hecho que poco a poco vayan apareciendo nuevos modelos de almacenamiento como NoSQL.

Toda base de datos se basa en 3 pilares fundamentales:

- **Consistencia:** toda consulta debe devolver los mismos datos, y en caso de error, antes el gestor optará por no devolver datos. Esto es importante en bases de datos donde las transacciones deben guardar siempre consistencia.
- **Disponibilidad:** toda consulta debe devolver algún dato, frente a posibles errores en el sistema, aunque estos no tengan por qué ser consistentes (pueden devolver diferentes salidas a dos clientes diferentes).
- **Tolerancia a partición:** los datos pueden guardarse distribuidamente en nodos.

A pesar de que son 3 pilares entre los que se mueve una base de datos, no es posible que los 3 se cumplan a la vez. Siempre van a ser mutuamente excluyentes. Esto significa que, en todos los casos solo puede priorizarse 2 pilares. Es por ello que, el teorema CAP se representa sobre un triángulo CAP y solo puede elegirse un vértice según necesidades particulares de caso.



Figura 5: Teorema CAP

Fuente. - Tomado de <https://dzone.com/refcardz/getting-started-nosql-and-data>

Las bases de datos relacionales se basan en el vértice de **CA**, es decir, proporcionan consistencia y disponibilidad, sin embargo, esto se consigue teniendo todos los datos en un mismo servidor, lo cual lo hace intolerante a particionamiento de los datos y, en consecuencia, no es escalable horizontalmente.

Las bases de datos no relacionales o NoSQL van a ser por definición escalables, y por tanto van a poseer la propiedad P. De este modo, si la escalabilidad horizontal es una necesidad, según las propiedades que necesitemos, tendremos que optar por una base de datos CP o AP.

Las **BBDD CP** priorizan consistencia, es decir, la respuesta va a ser siempre única. Pero esto se consigue a base de bloquear el sistema a consultas en caso de ocurrir 2 consultas simultáneas o un fallo, lo que da el riesgo de que una consulta no devuelva resultados. Estas soluciones van a ser ideales en transacciones económicas donde se prioriza que la transacción sea la correcta.

Las **BBDD AP** priorizan disponibilidad, es decir, en caso de una falla o nodo primario ocupado, el sistema ejecutará una de las réplicas en un nodo secundario. Esto significa que, siempre nos va a devolver resultados, pero existe un riesgo de que ese fallo (u operación simultánea) haya hecho perder información para que dos consultas den resultados diferentes. Estas soluciones serán ideales cuando busquemos una respuesta en todo momento, incluso si esta información no es 100% precisa como en redes sociales o sistemas que requieran consultas a tiempo real.

1.2.3. Características

Como hemos comentado anteriormente, el vacío que dejan las bases de datos relaciones en el triángulo CAP es solucionado por las bases de datos NoSQL y de ahí el objetivo de su creación y rápida adopción. La posibilidad de particionar los datos y ofrecer una escalabilidad horizontal en todo momento y tener la capacidad de gestionar datos heterogéneos, es decir, no responde a solo un esquema preestablecido es una clara ventaja. Las BBDD NoSQL basan sus propiedades en torno a este factor de escalabilidad horizontal y flexibilidad.

- **Esquemas flexibles:** no existe la estructura tabla, puede establecerse diferentes niveles jerárquicos, e incluso pudiendo guardarse diferentes tipos de datos en un mismo campo.
- **Sin esquemas predefinidos:** no es necesario predefinir el esquema de la base de datos, sino que se pueden añadir nuevos campos a posteriormente. Aun así, es recomendable tener una estructura predefinida por orden, aunque luego se cambie (esto se conoce como datos semiestructurados).
- **Escalabilidad horizontal:** permiten funcionamiento en cluster. No es necesario tener todos los datos en un mismo equipo, por lo que cuando se requiere más espacio o capacidad de procesamiento, solamente hay que añadir un servidor.
- **Replicabilidad y alta disponibilidad:** al funcionar en cluster, es posible tener varias copias de un mismo documento en varios equipos, de modo que, si uno de los equipos falla, automáticamente los otros toman el mando sin pérdidas de datos, ni rendimiento.
- **Particionado:** uno de los mayores problemas de almacenar grandes cantidades de datos es que no puede guardarse toda una base de datos en un solo documento debido al gran tamaño. La posibilidad de particionar se basa en que un documento puede distribuirse en varios equipos, eliminando la necesidad de que un solo disco tenga la capacidad de guardar todo un conjunto de datos.

- **Velocidad de consultas:** cuando las bases de datos crecen en tamaño, los tiempos de consulta de una SQL crecen, sobre todo cuando se requieren joins o relaciones en la consulta. En diferentes estructuras de datos y las arquitecturas en cluster reducen los tiempos de consulta cuando la cantidad de datos es de mayor tamaño.
- **Velocidad de procesado:** cuando se quieren hacer operaciones sobre toda la base de datos, las BBDD NoSQL llevan incorporados motores de procesamiento en paralelo, que reducen los tiempos de lectura y escritura.

Es importante remarcar que, aunque las bases de datos NoSQL giran generalmente en torno a estas propiedades, esto no significa que toda BBDD NoSQL vayan a cumplirlas, ni que todas las NoSQL son iguales de efectivas en estas propiedades. De hecho, las BBDD NoSQL se dividen en tipos, orientados a satisfacer unas prioridades u otras de acuerdo a diferentes necesidades que revisaremos en el próximo capítulo.

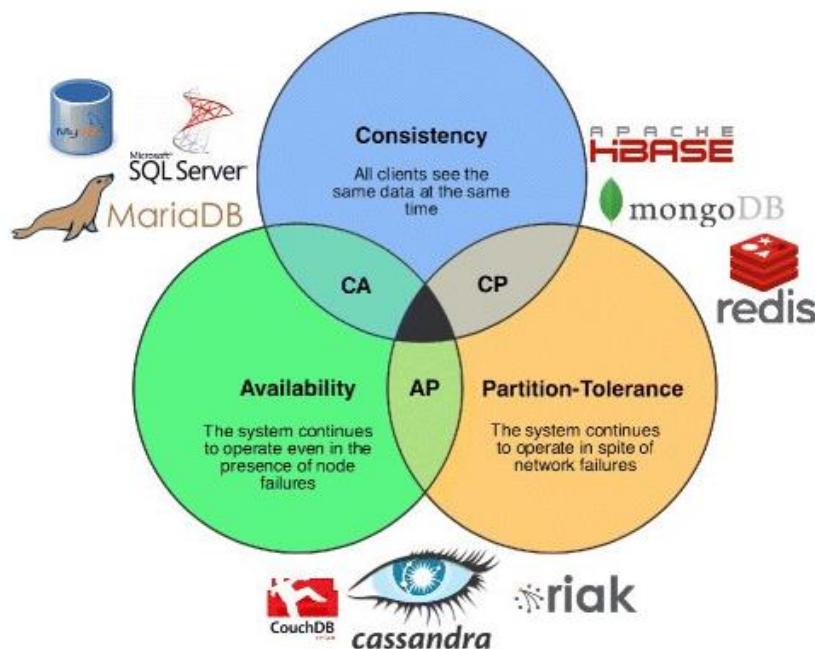


Figura 6: BBDD CAP NoSQL

Fuente. - Tomado de <https://gltaboada.github.io/tgdbook/conceptos-y-tipos-de-bases-de-datos-nosql-documental-columnar-clavevalor-y-de-grafos.html>

1.3. TIPOS DE BASES DE DATOS NOSQL

1.3.1. Clave/Valor

Esta modalidad de base de datos, key-value database o store en inglés, se basa en una tabla de tan solo dos columnas. En una de ellas se guarda un valor y en la otra, una clave que representa una característica identificativa única. Un valor puede ser sencillo, como una cadena de caracteres o un número entero, o pueden ser objetos complejos (un documento puede ocupar el lugar de un valor, aunque, entonces se hablaría de una base de datos de documentos). En las bases de datos se pueden incluir también referencias a archivos, así como a tuplas (conjunto de valores). El contenido de la base de datos puede ser muy heterogéneo, por lo que es posible incluir distintos objetos en una misma columna. En la mayoría de los casos, las claves seguirán un determinado esquema, pero esto no es indispensable. Tanto las cadenas de caracteres como los enteros se pueden constituir siguiendo criterios libres.

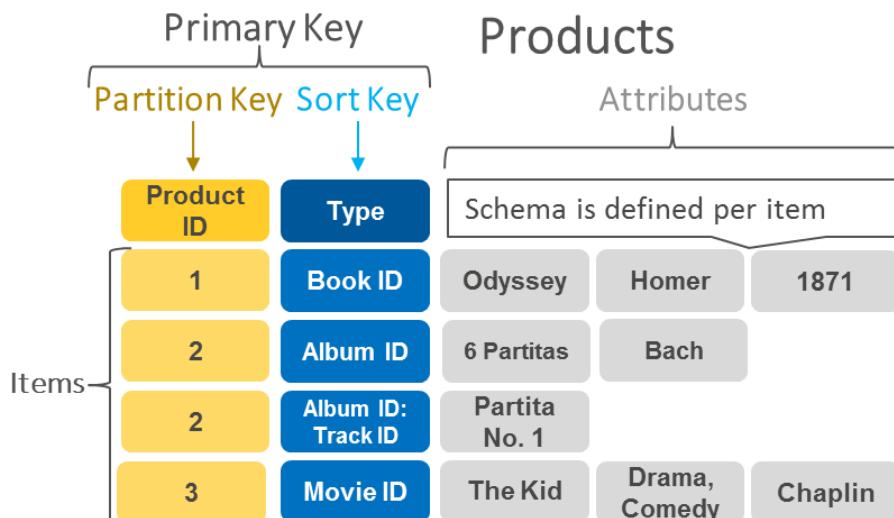


Figura 7: Diagrama de datos almacenados en DynamoDB de clave-valor
Fuente. - Tomado de <https://aws.amazon.com/es/nosql/key-value/>

Ventajas de las bases de datos clave-valor

Son efectivas en la consulta y fáciles de escalar, lo cual se deriva de la sencilla estructura del modelo. El almacenamiento **no exige ningún esquema** fijo, se pueden realizar modificaciones sobre la marcha, de modo que es posible añadir nuevos campos mientras se realizan acciones en otras entradas.

Asimismo, ofrece una gran velocidad de búsqueda gracias a la **sencilla conexión entre la clave y el valor**. Si se desea recuperar información, solo se tendrá que consultar un valor a través de una clave específica, esto constituye al mismo tiempo una desventaja de este tipo de almacenamiento, ya que no contempla otro método de acceso como en la base de datos relacionales.

En el almacenamiento de clave-valor únicamente está prevista la búsqueda a través de la clave, por lo que, en general, se debe renunciar a otras indicaciones o métodos de búsqueda, priorizando la velocidad con volumen de datos.

Bases de datos clave-valor

- Amazon DynamoDB
- Berkeley DB
- Redis
- Riak
- Voldemort

1.3.2. Documentales

Las bases de datos documentales están diseñadas para almacenar documentos en los formatos JSON o XML. A diferencia de las bases de datos relacionales, el esquema para cada documento puede variar, lo cual ofrece a los desarrolladores más flexibilidad en la organización y almacenamiento de datos de las aplicaciones, así como una reducción del almacenamiento en valores opcionales. La naturaleza flexible, semiestructurada y jerárquica de documentos y bases de datos de documentos permite que evolucionen según las necesidades de las aplicaciones.

```
JSON
1  [
2    {
3      "year" : 2013,
4      "title" : "Turn It Down, Or Else!",
5      "info" : {
6          "directors" : [ "Alice Smith", "Bob Jones"],
7          "release_date" : "2013-01-18T00:00:00Z",
8          "rating" : 6.2,
9          "genres" : [ "Comedy", "Drama"],
10         "image_url" : "http://ia.media-imdb.com/images/N/09ERWAU7FS797AJ7LU8HN09AMUP908RLlo5JF90EWR7LJKQ7@.V1_SX400_.jpg",
11         "plot" : "A rock band plays their music at high volumes, annoying the neighbors.",
12         "actors" : [ "David Matthewman", "Jonathan G. Neff"]
13     }
14   },
15   {
16     "year": 2015,
17     "title": "The Big New Movie",
18     "info": {
19       "plot": "Nothing happens at all.",
20       "rating": 0
21     }
22   }
23 ]
```

Figura 8: Documento de tipo JSON
Fuente. - Tomado de <https://aws.amazon.com/es/nosql/document/>

Ventajas de las bases de documentales

Es una opción para aplicaciones de administración de contenido, como blogs y plataformas de video. Es más intuitiva para que un desarrollador actualice una aplicación a medida que evolucionan los requisitos. Además, si el modelo de datos necesita cambiar, solo se deben actualizar los documentos afectados. No se requiere actualización del esquema y no es necesario tiempo de inactividad de la base de datos para realizar los cambios.

Son eficientes y efectivas para almacenar información de catálogo. Por ejemplo, una aplicación de e-commerce, los diferentes productos generalmente tienen diferentes números de atributos. La administración de miles de atributos en bases de datos relacionales no es eficiente y afecta al rendimiento de lectura. Al utilizar una base de datos de documentos, los atributos de cada producto se pueden describir en un solo documento para que la administración sea fácil y la velocidad de lectura sea más rápida. El cambio de atributos de un producto no afectará a otros.

Bases de datos documentales

- MongoDB
- DynamoDB
- Couchbase
- Azure Cosmos
- RavenDB

1.3.3. Grafos

En este tipo de bases de datos, la información se representa como nodos de un grafo y sus relaciones con las aristas de este, de manera que se puede hacer uso de la teoría de grafos para recorrerla. Para sacar el máximo rendimiento a este tipo de bases de datos, su estructura debe estar totalmente normalizada, de forma que cada tabla tenga una sola columna y cada relación dos.

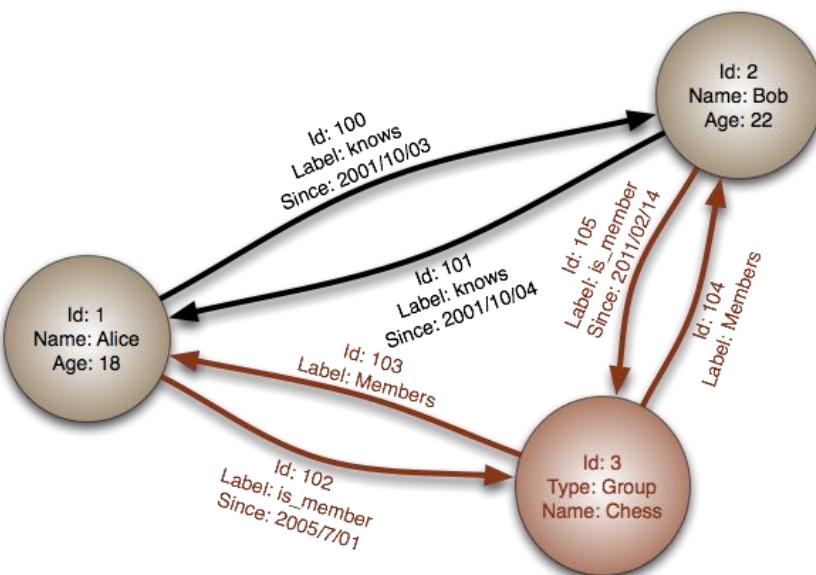


Figura 9: GraphDatabase

Fuente. - Tomado de https://es.m.wikipedia.org/wiki/archivo:graphdatabase_propertygraph.png

Ventajas de las bases de grafos

Son las más recomendables cuando tenemos datos altamente interconectados y necesitamos recorrer relaciones de referencias. En general, son ideales para trabajar con datos altamente interconectados.

- Relaciones sociales
- Recursos humanos
- Policía
- Redes de sistemas
- Detección del fraude
- Estructuras organizativas de empresas
- Cálculo de rutas en logística

Bases de datos grafos

- AllegroGraph
- ArangoDB
- Neo4j
- Cytoscape
- GraphBase

1.3.4. Columnas anchas

Las bases de datos NoSQL de columnas anchas almacenan datos en tablas con filas y columnas similares a las BBDD relacionales, pero los nombres y los formatos de las columnas pueden variar de fila a fila en la tabla. Las columnas de columnas anchas agrupan columnas de datos relacionados juntos. Una consulta puede recuperar datos relacionados en una sola operación porque sólo se recuperan las columnas asociadas con la consulta. En una BBDD los datos estarían en diferentes filas y almacenadas en diferentes lugares del disco, requiriendo operaciones múltiples para su recuperación.

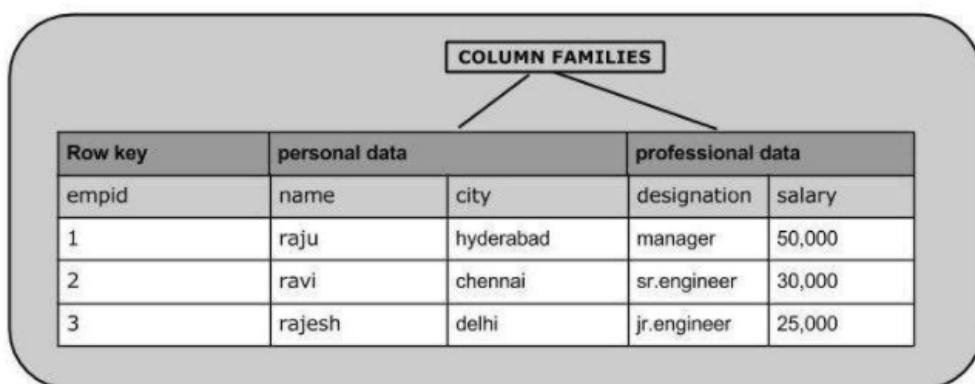


Figura 10: Esquema de Almacenamiento

Fuente. - Tomado de <https://www.tecnologias-informacion.com/columnasanchas.html>

Ventajas de las bases de columnas anchas

Las bases de datos columnares hacen alarde de sus ventajas cuando se deben realizar operaciones de evaluación de grandes volúmenes de datos. Dado que el acceso al disco duro suele ser el cuello de botella en la lectura de cualquier base de datos y el acceso a una base de datos columnar es más eficiente. Aquí radica la mayor ventaja de la variante basada en columnas.

Otra ventaja es la de **compresión**. Los datos de una columna son siempre del mismo tipo, por ejemplo, una cadena o un entero. Como todas las entradas de un tipo están próximas entre sí, se pueden comprimir de forma más eficiente.

Bases de datos grafos columnas anchas

- Amazon Redshift
- MariaDB-ColumnStore
- Apache Cassandra
- MonetDB
- SAP HANA

1.3.5. Ejemplos de bases de datos NoSQL

- **MongoDB:** es un sistema de base de datos NoSQL orientado a documentos de código abierto y escrito en C++, que en lugar de guardar los datos en tablas lo hace en estructuras de datos con un esquema dinámico. Al ser un proyecto de código abierto, sus binarios están disponibles para los sistemas operativos Windows, Linux, OS y Solaris. (Robledano, 2019). Al estar escrito en C++, cuenta con una capacidad para aprovechar los recursos de máquina.



Figura 11: MongoDB
Fuente. - Tomado de <https://www.mongodb.com/es>

- **Redis:** significa Remote Dictionary Server (servidor de diccionarios remoto), es un almacén de datos clave-valor en memoria de código abierto que se puede utilizar como base de datos, caché, agente de mensajes y cola. Redis ofrece tiempos de respuesta inferiores al milisegundo, lo que permite que se realicen millones de solicitudes por segundo para aplicaciones en tiempo real de videojuegos, publicidad, banca, salud e IoT.



Figura 12: Redis
Fuente. - Tomado de <https://redislabs.com/>

- **Cassandra:** software que fue lanzado en 2008, originalmente desarrollado por Facebook, posteriormente fue dado a la fundación Apache, hoy en día es una herramienta de código abierto usada por gigantes de la informática como Twitter. Cassandra permite el manejo masivo de datos, de una forma escalable, de hecho, esta es su mayor virtud, la capacidad lineal de escalar que posee, es decir, si tenemos 2 nodos podemos realizar 1000 operaciones por segundo, si tuviésemos 4 nodos estaríamos el doble de operaciones, así sucesivamente.



Figura 13: Cassandra
Fuente. - Tomado de https://cassandra.apache.org/_index.html

- **CouchDB:** base de datos de código abierto desarrollada por la fundación de software Apache. Se centra en la facilidad de uso, abarcando la web. Es una base de datos del almacén de documentos. Utiliza JSON para almacenar los datos (documentos), script java como lenguaje de consulta para transformar los documentos, protocolo HTTP API para acceder a documentos y consultar los índices en el navegador web. CouchDB incluye una **API REST** basada en HTTP, que ayuda a comunicarse con la base de datos de manera directa.



Figura 14: CouchDB
Fuente. - Tomado de <http://couchdb.apache.org/>

- **Neo4j:** es un software libre de base de datos orientada a grafos, creada por Neo Technology en Java. Neo4j almacena datos estructurados en grafos en lugar de en tablas, es decir, la data se almacena de forma relacionada formando un grafo dirigido entre los nodos y las relaciones entre ellos.



Figura 15: CouchDB
Fuente. - Tomado de <https://neo4j.com/>

1.3.6. Ventajas y desventajas

Las bases de datos de NoSQL presentan muchas ventajas en comparación con las bases de datos tradicionales:

- **Versatilidad:** la ventaja de esta nueva tecnología difiere de las demás soluciones de bases de datos es la versatilidad que ofrece a crecimientos o cambios sobre la forma cómo almacena la información, si fuera necesario agregar un nuevo campo sobre una “colección” (en una base de datos relacional sería una tabla), dado que se basan sobre una notación de intercambio de documentos JSON, simplemente se adiciona al documento y todo sigue operando, configuraciones extras.

- **Crecimiento horizontal:** soporta la escalabilidad descentralizada, es decir, permite las estructuras distribuidas, si durante la operación se ve que el desempeño de uno de los servidores es bajo, se instalan nuevos nodos operativos para que balanceen la carga de trabajo y a eso se le denomina crecimiento horizontal.
- **Disponibilidad de recursos:** no se requieren granjas de servidores con grandes cantidades de recursos para operar, pueden empezar a operar con bajos recursos e ir creciendo, dependiendo de las necesidades sin tener que detener los servicios de operación.
- **Optimización:** los sistemas NoSQL tienen un algoritmo interno para reescribir las consultas escritas por los usuarios o las aplicaciones programadas, esto con el fin de no sobrecargar el rendimiento de los servidores y mantener un nivel óptimo en las operaciones.

Las bases de datos de NoSQL también presentan desventajas en comparación con las bases de datos tradicionales:

- **Atomicidad:** no todas las bases de datos tienen características de la atomicidad en la información, es decir, que la información en ocasiones no es consistente, puede ser diferente en cada uno de los nodos que se puedan configurar la arquitectura de base de datos.
- **Documentación del software:** NoSQL es nuevo, operaciones pueden ser limitadas y se requiere de conocimientos avanzados sobre el uso de las herramientas y las personas que desarrollen deben invertir más tiempo en los desarrollos.
- **Estándares en el lenguaje:** no se tiene un estándar definido entre los diferentes motores que ofrecen este servicio, es decir, por ejemplo: DB2 para poder insertar información sobre su base de datos, el manejo de los objetos JSON no es el mismo como se utiliza en MongoDB y con ellos la diversidad de conocimientos que se debe tener dependiendo de la solución NoSQL se vaya a utilizar.
- **Herramientas GUI (Graphical User Interface):** las herramientas que ofrecen para la administración de estas herramientas tienen acceso por consola, no tienen una interfaz gráfica y requiere de conocimiento de comandos para su mantenimiento.

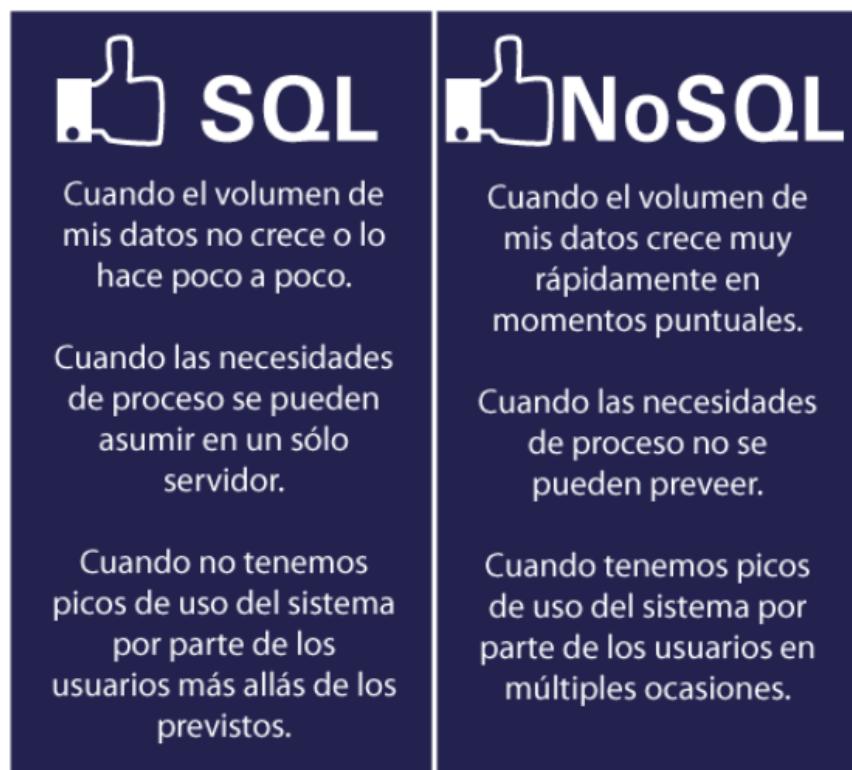


Figura 16: SQL o NoSQL

Fuente. - Tomado de <https://www.pragma.com.co/academia/lecciones/bases-de-datos-relacionales-vs.-no-relacionales>

1.4. SQL VS NOSQL

1.4.1. Cargas de trabajo

- Las bases de datos relacionales están diseñadas para ofrecer el soporte a las transacciones (OLTP) y procesamiento analítico (OLAP) en línea, resultado de las operaciones de SELECT, INSERT, DELETE y UPDATE.
- Las bases de datos no relacionales están diseñadas para varios patrones de acceso a los datos de baja latencia, es decir, su enfoque es de análisis de los datos semiestructurados a diferencia de la estructurada en la base de datos relaciones.
- Cabe señalar que, la adopción y éxito de la adopción de cualquiera de estas tecnologías debe centrarse en el objetivo del proyecto a implementar. Eso quiere decir, la ventaja técnica se logra en base al escenario de aplicación.

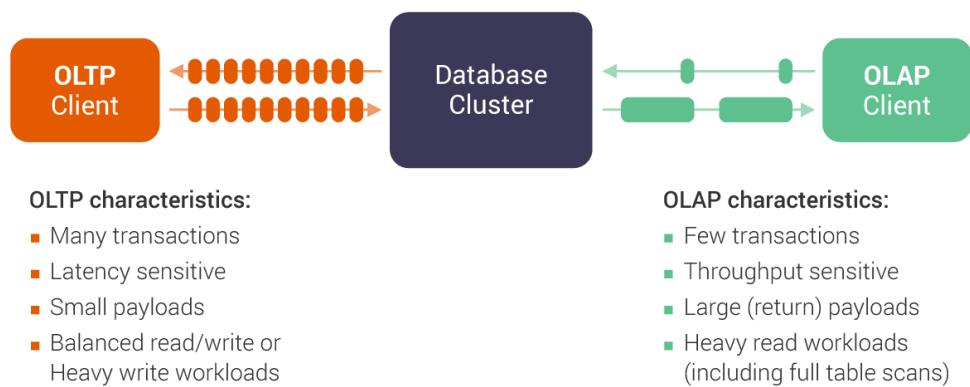


Figura 17: OLAP vs OLTP
Fuente. - Tomado de <https://hackr.io/blog/olap-vs-oltp>

1.4.2. Modelado de datos

- Las bases de datos relacionales el modelo relacional normaliza los datos en tablas. A partir de esta definición se establecen las filas, columnas, índices y relaciones como elementos centrales de integridad referencial.
- Las bases de datos no relacionales se proporciona una variedad de modelos de datos que están optimizados para el rendimiento y la escala, es decir, las bases de datos NoSQL soporta 4 formas especializadas de funcionamiento a diferencia de las relacionales que solo permite el normalizado.

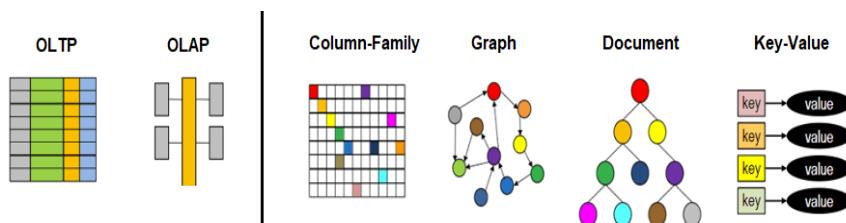


Figura 18: Modelos de Gestión de Datos (SQL vs NoSQL)
Fuente. - Tomado de <https://www.sqlindia.com/what-is-nosql-and-how-it-is-different-than-sql/>

1.4.3. Propiedades ACID

- Las bases de datos relacionales brindan atomicidad, coherencia, aislamiento y durabilidad (ACID):
 - **Atomicidad** requiere que una transacción se ejecute por completo o no se ejecute en absoluto.
 - **Coherencia** requiere la confirmación de las transacciones, es decir, que los datos se acoplen al esquema de la base de datos.
 - **Aislamiento** requiere que las transacciones simultáneas se ejecuten por separado.
 - **Durabilidad** requiere de capacidades de recuperación por fallas del sistema o corte de energía y volver al último estado conocido.
- Las bases de datos no relacionales hacen concesiones al flexibilizar algunas de las propiedades ACID debido al modelo de datos flexible. Esto hace que las bases de datos NoSQL sean una excelente opción en escenarios de baja latencia y alto rendimiento para crecimiento escalar horizontal.

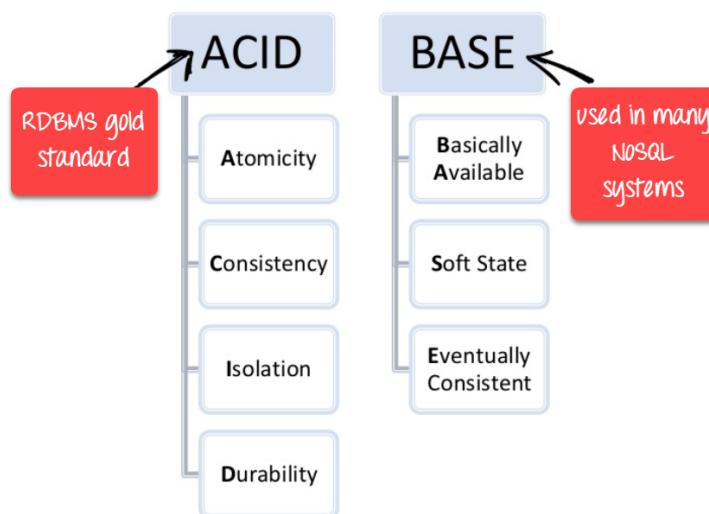


Figura 19: ACID vs BASE
Fuente. - Tomado de <https://www.guru99.com/sql-vs-nosql.html>

1.4.4. Rendimiento

- Las bases de datos relacionales dependen de un subsistema de disco. Esto se debe a que este enfoque necesita la optimización de las consultas, índices y estructura de tabla para lograr el máximo rendimiento.
- En las bases de datos no relacionales el rendimiento se asocia al tamaño del clúster de hardware, la latencia de red y aplicación.

1.4.5. Escalado

- Las bases de datos relacionales escalan sus capacidades de computación del hardware o ampliación por réplicas de cargas de trabajo para lecturas. Este escenario evita la congestión en soluciones OLTP y OLAP separándolas en 2 instancias para disminuir el tráfico y no afecte al ambiente de producción.
- Las bases de datos no relacionales pueden particionarse mediante el uso de arquitectura distribuida para aumentar su rendimiento porque sus patrones de acceso son escalables.

1.4.6. Terminologías

La siguiente tabla compara las terminologías utilizadas por las bases de datos NoSQL seleccionadas con la terminología utilizada por las bases de datos SQL.

Tabla 1
Terminologías

SQL	MongoDB	DynamoDB	Cassandra
Tabla	Colección	Tabla	Tabla
Fila	Documento	Elemento	Fila
Columna	Campo	Atributo	Columna
Clave principal	ObjectId	Clave principal	Clave principal
Índice	Índice	Índice secundario	Índice

Nota. Elaboración Propia

Resumen

1. Un SGBD es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información de una base de datos.
2. Las bases de datos NoSQL no cumplen con el esquema entidad-relación.
3. El teorema CAP dice que en sistemas distribuidos (NoSQL) es imposible garantizar a la vez:
 - Consistencia
 - Disponibilidad
 - Tolerancia a particiones

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://www.youtube.com/watch?v=yJ4IGdXQUGA>
- <https://www.youtube.com/watch?v=knVwokXITGI>
- <https://www.youtube.com/watch?v=0xx3fcYcb28>



MONGODB

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno identifica los componentes y herramientas para la integración de información del cliente desde múltiples canales.

TEMARIO

2.1 Tema 5 : ¿Qué es MongoDB?

- 2.1.1 : Introducción
- 2.1.2 : ¿Por qué utilizar MongoDB?
- 2.1.3 : Ventajas y desventajas
- 2.1.4 : Características
- 2.1.5 : Arquitectura
- 2.1.6 : Estructura

2.1 Tema 6 : Entorno de trabajo

- 2.2.1 : Herramientas
- 2.2.2 : Instalación
- 2.2.3 : Configuración
- 2.2.4 : Consola de trabajo
- 2.2.5 : Tipos de datos

ACTIVIDADES PROPUESTAS

- Los alumnos realizan las lecturas propuestas.
- Los alumnos realizan la instalación de la base de datos MongoDB.

2.1. ¿QUÉ ES MONGODB?

2.1.1. Introducción

Es una base de datos NoSQL (no relacional) de código abierto. Este tipo de soluciones se basan en el principio de almacenar datos en estructuras tipo llave-valor. MongoDB se enfoca específicamente en que los valores de estas llaves (colecciones) son estructuras tipo JSON (llamados documentos), es decir objetos JavaScript (lenguaje sobre el cual se basa la solución).

MongoDB posee varias estrategias de manejo de datos que la han posicionado donde se encuentra hoy en día, tales como, sus procesos de división de datos en distintos equipos físicos (clusterización). En caso similar, documentos muy grandes que superen un límite estipulado aplica una estrategia que divide y los almacena por separado. Al solicitarlos, el driver se encarga de armar automáticamente.

2.1.2. ¿Por qué utilizar MongoDB?

La gestión de la información se vuelve más complejo día a día. NoSQL es la respuesta a todos estos problemas, no es un sistema de gestión tradicional, ni siquiera un sistema de gestión de bases de datos relacionales (RDBMS). NoSQL es un tipo de base de datos que puede manejar y ordenar datos desestructurados, desordenados y complicados.

MongoDB representa un cambio radical y muy necesario con respecto a las tecnologías de bases de datos relacionales, es una seria alternativa complementaria que se adapta muy bien a las nuevas necesidades de gestión de información con un alto rendimiento y escalabilidad en distintos escenarios.



Figura 20: Gartner Database
Fuente. - Tomado de <https://www.gartner.com/en>

2.1.3. Ventajas y desventajas

Las ventajas más importantes que ofrece:

- **Plataforma de datos distribuidos:** se puede ejecutar en todos los centros de datos distribuidos para garantizar nuevos niveles de disponibilidad y escalabilidad.
- **Modelo de datos flexible:** permite el almacenamiento de los datos en documentos flexibles, lo que hace que la persistencia de los datos y la combinación sean fáciles.
- **Desarrollo rápido e iterativo:** modelo de datos simple y esquema dinámico. Tiene una interfaz gráfica de usuario y herramientas de línea de comando que facilitan a los desarrolladores la creación y evolución de aplicaciones.
- **TCO reducido:** equipo de operaciones puede realizar su trabajo en nube. Los costos se reducen significativamente ya que su despliegue ejecuta hardware básico.
- **Conjunto de características integrado:** se pueden obtener diversas aplicaciones en tiempo real gracias al análisis y visualización de datos, canalizaciones de los datos de transmisión por eventos, búsquedas de textos y geoespacial, procesamiento de imágenes, replicación global de manera confiable y segura.
- **Comunidad:** tiene una comunidad activa que brinda soporte y conocimiento.

Las desventajas más importantes que tiene:

- **Sentencias SQL:** no admiten el 100% de las consultas existentes.
- **Interfaces gráficas:** la única opción para trabajar con MongoDB sea por consola de comandos.
- **Falta de estandarización entre las diferentes bases de datos:** este punto va más por la adopción de un esquema común dentro del ámbito de bases de datos NoSQL y debemos considerar que el mercado todavía es muy nuevo.

Esta lista de ventajas y desventajas puedan ayudar a definir el tipo de bases de datos a tomar en cuenta en futuras implementaciones, sin embargo, la correcta elección debe de asociarse primero a las necesidades del proyecto, y no a la tecnología. Por ejemplo: si estimamos implementar un proyecto Big Data, MongoDB tiene una alta probabilidad de ser confiable, sin embargo, si queremos desarrollar una ERP no es la mejor opción.

2.1.4. Características

- **Consultas AD HOC:** soporta consultas por campos, rangos y expresiones regulares y puede devolver un campo específico del documento o una función JavaScript.
- **Indexación:** puede indexar cualquier campo del documento, así como crear índices secundarios que brindar rapidez y soporte a las consultas.
- **Replicación:** ofrece un soporte a réplicas de tipo maestro-esclavo, de modo que el maestro puede realizar las lecturas y las escrituras mientras que el esclavo copia la

información almacenada en el maestro y sólo puede usarse para operaciones de lectura o copias de seguridad.

- **Balanceo de carga:** puede escalar horizontalmente mediante llaves shard. De este modo, un shard es un maestro con esclavos y los datos son distribuidos por rangos entre las instancias de la base de datos.
- **Almacenamiento de archivos:** puede utilizarse con sistema de archivos, tomando como ventaja de la capacidad que tiene MongoDB para el balanceo de carga y la replicación de datos utilizando múltiples servidores. (Alonso, 2021).
- **Agregación:** posee el concepto de MapReduce para el procesamiento por lotes de datos y operaciones de agregación. Esta función es similar al “group-by” de SQL relacional. Ideal para incrementar el performance de la solución.
- **Ejecución de JavaScript:** puede realizarse consultas JavaScript del lado del servidor de modo que estas son ejecutadas directamente sobre la base de datos.

2.1.5. Arquitectura

MongoDB está compuesto por las siguientes capas principales:

- **MongoDB Query Language (MQL) o Lenguaje de Consulta:** lenguaje utilizado para interactuar con MongoDB. Esta capa se encarga de gestionar todas las operaciones CRUD que se efectúan sobre la base de datos. Se encarga de traducir las peticiones efectuadas desde el driver de la aplicación cliente mediante mensajes BSON.
- **Modelo de Datos de Documentos (Document Data Model):** se encarga de aplicar las instrucciones de la capa anterior sobre las estructuras de datos. Se encarga por tanto de manejar las estructuras de datos, espacios de nombres e índices.
- **Capa de almacenamiento (Storage Layer):** se encarga de la persistencia de los datos en disco. Maneja cuestiones como las llamadas al sistema, los accesos a disco, los sistemas de compresión y las estructuras de archivos.

MongoDB permite la creación de clúster de servidores para facilitar soluciones de alta disponibilidad, en los que existe un servidor principal, y varios secundarios que actúan como réplicas de información. Al servidor principal se le conoce como nodo primario, y es responsable de procesar las peticiones de entrada y salida. Los demás servidores se les denomina secundarios, y tienen como misión de almacenar copias redundantes de la información contenida en el nodo primario. MongoDB implementa el protocolo Raft, que se encarga de activar automáticamente uno de los nodos secundarios a modo del nodo principal en caso de que el nodo principal falle, sin caída del servicio.

Para ello, se utiliza el concepto de fragmento (shard). Se puede dividir la base de datos en varios nodos shard, y pueden añadir nodos shard en caso de necesidad para ampliar la base de datos. El componente MongoS es el encargado del enruteado de información hacia los nodos shard implicados en la operación.

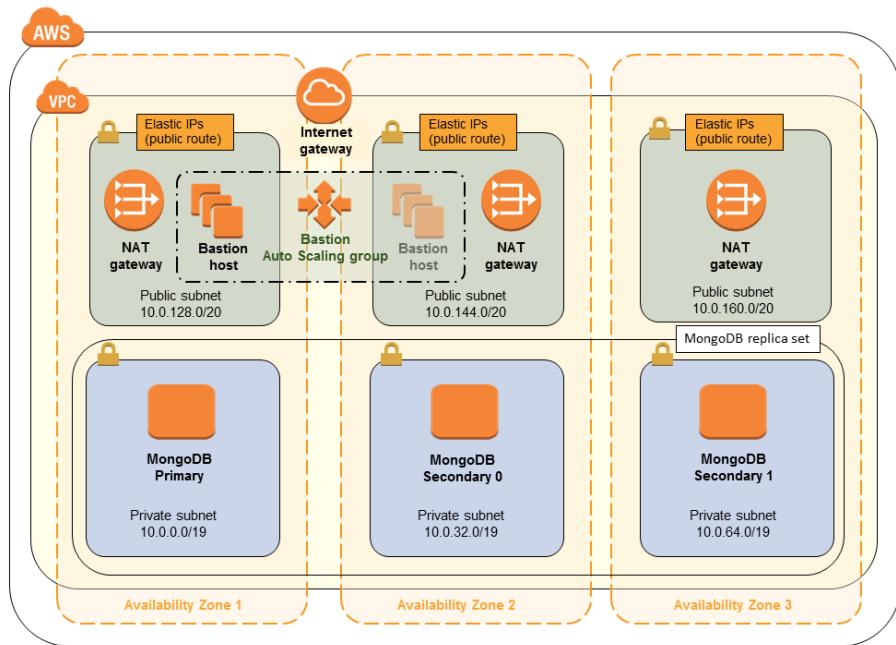


Figura 21: MongoDB in AWS Platform

Fuente. - Tomado de https://docs.aws.amazon.com/es_es/quickstart/latest/mongodb/architecture.html

2.1.6. Estructura

Cada registro o conjunto de datos se denomina documento. Pueden agruparse en colecciones, equivalente a tablas de las bases de datos relacionales, sin estar sujetos a un esquema fijo de tipo tradicional. La jerarquía en MongoDB a diferencia de los sistemas gestores de bases de datos relacionales, no se utiliza tablas, ni filas ni columnas, sino documentos con estructuras.

En ese contexto, un conjunto de campos formará un documento, que en caso de asociarse con otros formará una colección. Las bases de datos estarán formadas por grupos de colecciones, y a su vez, cada servidor puede tener tantas bases de datos como el equipo físico permita. En el caso del intercambio y transferencia de documentos se utiliza BSON que ya hemos detallado en capítulos anteriores.

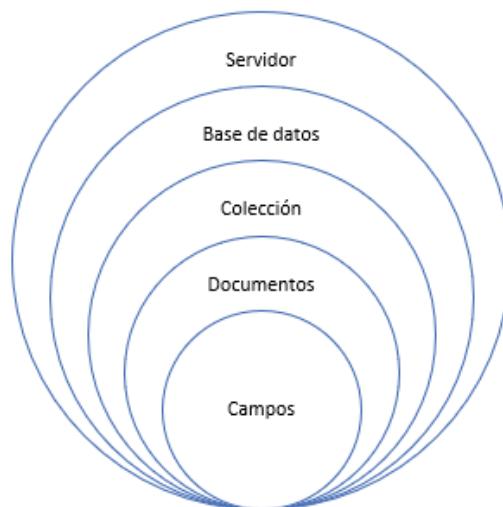


Figura 22: Jerarquía de Objetos MongoDB

Fuente . - Elaboración Propia

2.2. ENTORNO DE TRABAJO

2.2.1. Herramientas

MongoDB incluye distintas herramientas que permiten trabajar con las bases de datos desde diferentes perspectivas y propósitos:

- **MongoDB Database Tools:** colección de utilidades de línea de comandos para trabajar con una implementación de MongoDB.

Importación / exportación binaria

- Mongodump: exportación del contenido de una base de datos.
- Mongorestore: restaura los datos desde un mongodump.
- Bsondump: Convierte archivos de volcado BSON en JSON

Importación / exportación de datos

- Mongoimport: importa el contenido desde un archivo de exportación JSON, CSV o TSV extendido.
- Mongoexport: produce una exportación en formato JSON o CSV de los datos almacenados en una instancia.

Herramientas diagnósticas

- Mongostat: proporciona una descripción del estado de una instancia en ejecución.
- Mongotop: proporciona una descripción del tiempo que una dedica a leer y escribir datos una instancia.

Herramientas GridFS

- Mongofiles: admite la manipulación de archivos almacenados en su instancia de MongoDB en objetos GridFS.
- **MongoDB Connector BI:** facilita la creación de visualizaciones de datos con herramientas de análisis populares, es decir, las herramientas tradicionales de inteligencia empresarial están diseñadas para trabajar con datos tabulares, de filas y columnas. El conector permite consultar datos de MongoDB con SQL mediante herramientas como Tableau, Power BI y Excel.

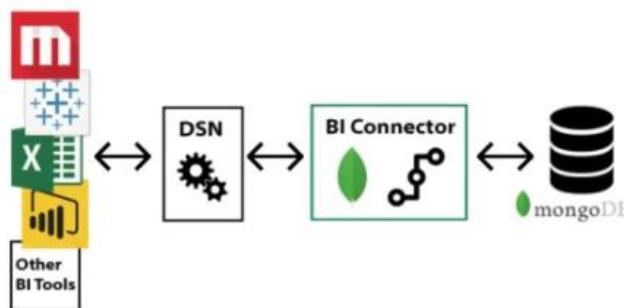


Figura 23: MongoDB Connector for BI

Fuente. - Tomado de <https://docs.mongodb.com/bi-connector/upcoming/>

- **MongoDB Compass:** es una GUI para consultar, agregar y analizar sus datos de en un entorno visual.
- **Importación de datos:** importar datos de archivos CSV o JSON a su base de datos MongoDB.
- **Consulta de datos:** escribir consultas ad-hoc para filtrar sus datos.
- **Creación de canalizaciones de agregación:** permite que documentos de una colección o vista pasen por varias etapas de procesado en un conjunto de resultados agrupados.
- **Ejecutar comandos en el Shell:** controlar la salida de los datos en un entorno interactivo JavaScript.
- **MongoDB Shell:** es un completo y funcional entorno JavaScript y Node.js 14.x que permite interactuar con despliegues MongoDB. Puede usarse en consultas y operaciones directamente con su base de datos.

The screenshot shows the MongoDB Shell interface. At the top, there is a header bar with a warning icon and the text 'EJEMPLO'. Below this, a message says 'Ejecute el siguiente comando dentro de MongoDB Shell :'. A code editor window contains the following JavaScript code:

```
// Entering editor mode (^D to finish, ^C to cancel)
const { Writable } = require('stream');

const myWritable = new Writable({
  write(chunk, encoding, callback) {
    // ...
  },
  writev(chunks, callback) {
    // ...
  }
});
```

Figura 24 MongoDB Shell

Fuente. - Tomado de <https://docs.mongodb.com/mongodb-shell/>

- MongoDB Charts:** es una herramienta para crear representaciones visuales de sus datos MongoDB que hace que la comunicación de sus datos sea un proceso sencillo al proporcionar herramientas integradas para compartir fácilmente los datos con visualizaciones dinámicas.

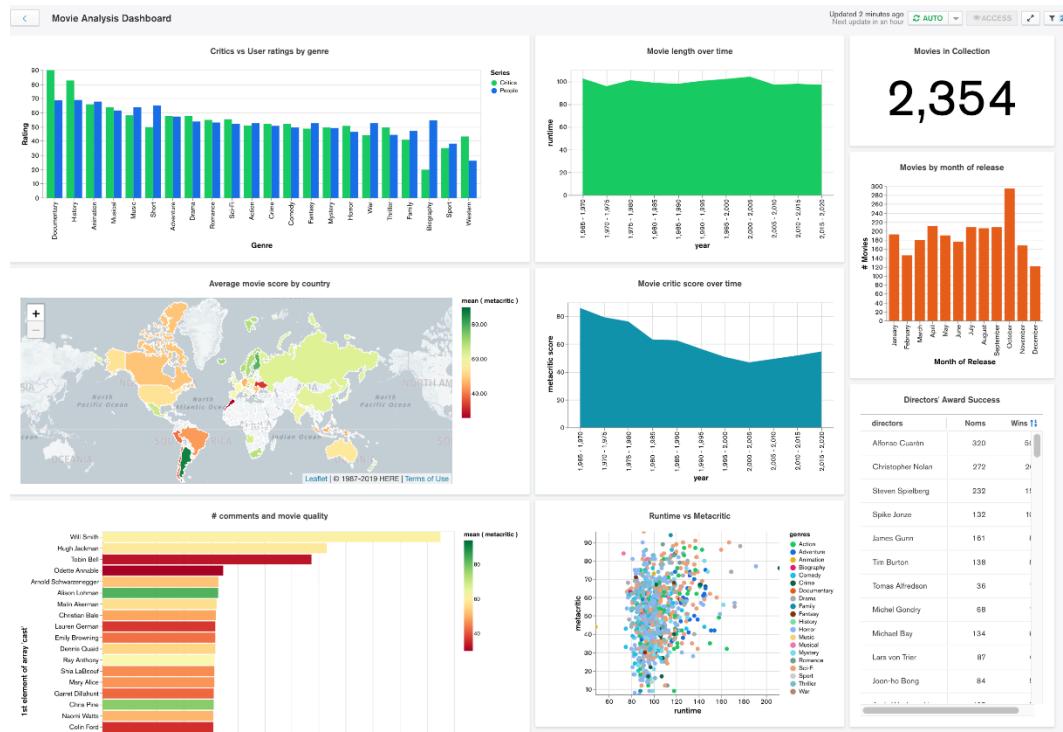


Figura 25 MongoDB Chart

Fuente. - Tomado de <https://docs.mongodb.com/charts/saas/>

- MongoDB Command Line Interface (CLI):** permite la administración desde línea de comando la interacción con MongoDB Atlas, Cloud Manager y Ops Manager para automatizar tareas.

```

~ mongocli config
You are configuring a profile for mongocli.
All values are optional and you can use environment variables (MCLI_*) instead.
Enter [?] or any option to get help.

? Public API Key: abcdef
? Private API Key: [? for help] *****
? Choose a default organization: Org1 (5e39bf1212121e685774c81c)
? Choose a default project: Project3 (5e39bf4979358e6857741212)

Your profile is now configured.
You can use [mongocli config set] to change these settings at a later
time.

```

Figura 26: MongoDB CLI

Fuente. - Tomado de <https://docs.mongodb.com/mongocli/stable/>

2.2.2. Instalación

MongoDB está disponible en dos ediciones de servidor Community y Enterprise. Existe un servicio en nube de MongoDB que no requiere de gastos de instalación que pueden evaluarse.

Paso 01: accede a la página oficial de MongoDB.

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/>

Install MongoDB > Install MongoDB Community Edition

Install MongoDB Community Edition on Windows

NOTE

MongoDB Atlas

MongoDB Atlas is a hosted MongoDB service option in the cloud which requires no installation overhead and offers a free tier to get started.

On this page

- Overview
- Considerations
- Install MongoDB Community Edition
- Run MongoDB Community Edition as a Windows Service
- Run MongoDB Community Edition from the Command Interpreter
- Additional Considerations

Overview

Use this tutorial to install MongoDB 5.0 Community Edition on Windows using the default installation wizard.

MongoDB Version

This tutorial installs MongoDB 5.0 Community Edition. To install a different version of MongoDB Community, use the version drop-down menu in the upper-left corner of this page to select the documentation for that version.

Figura 27: MongoDB Install Procedure

Fuente. - Tomado de <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/#install-mongodb-community-edition>

Paso 02: clic en la opción “Install MongoDB Community Edition”.

Install MongoDB > Install MongoDB Community Edition

Install MongoDB Community Edition on Windows

NOTE

MongoDB Atlas

MongoDB Atlas is a hosted MongoDB service option in the cloud which requires no installation overhead and offers a free tier to get started.

On this page

- Overview
- Considerations
- Install MongoDB Community Edition**
- Run MongoDB Community Edition as a Windows Service
- Run MongoDB Community Edition from the Command Interpreter
- Additional Considerations

Overview

Use this tutorial to install MongoDB 5.0 Community Edition on Windows using the default installation wizard.

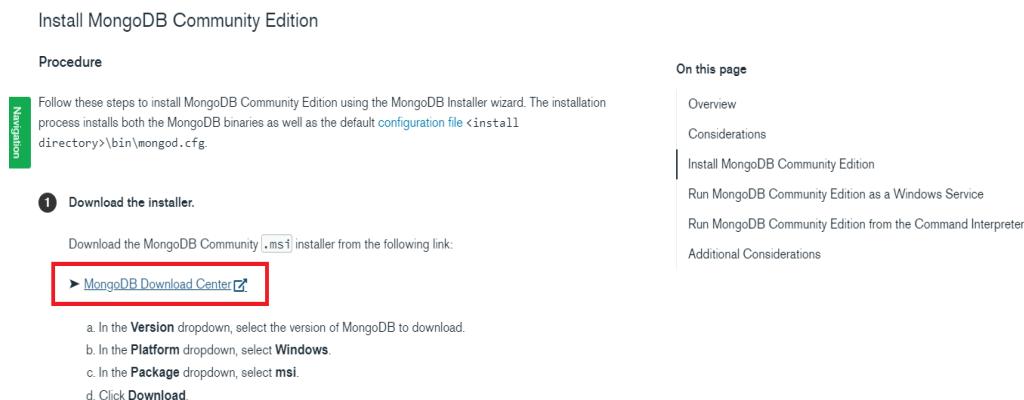
MongoDB Version

This tutorial installs MongoDB 5.0 Community Edition. To install a different version of MongoDB Community, use the version drop-down menu in the upper-left corner of this page to select the documentation for that version.

Figura 28: MongoDB Install Procedure

Fuente. - Tomado de <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/#install-mongodb-community-edition>

Paso 03: clic en la opción “MongoDB Download Center”.



Install MongoDB Community Edition

Procedure

Follow these steps to install MongoDB Community Edition using the MongoDB Installer wizard. The installation process installs both the MongoDB binaries as well as the default configuration file <install directory>\bin\mongod.cfg.

- Download the installer.**

Download the MongoDB Community .msi installer from the following link:

[► MongoDB Download Center](#)

a. In the **Version** dropdown, select the version of MongoDB to download.
b. In the **Platform** dropdown, select **Windows**.
c. In the **Package** dropdown, select **msi**.
d. Click **Download**.

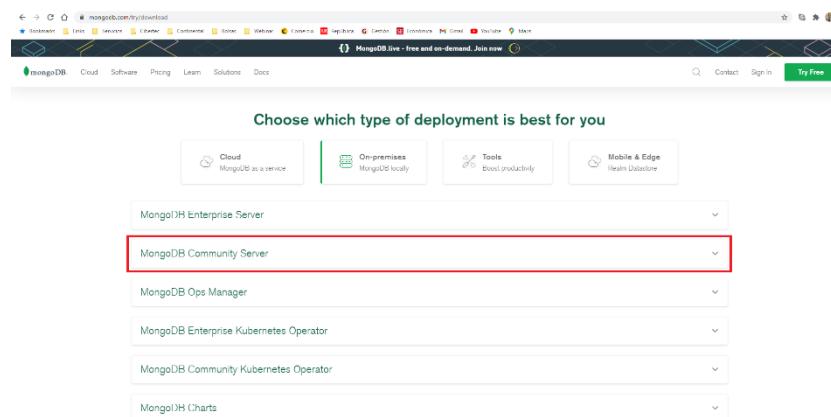
On this page

- Overview
- Considerations
- Install MongoDB Community Edition
- Run MongoDB Community Edition as a Windows Service
- Run MongoDB Community Edition from the Command Interpreter
- Additional Considerations

Figura 29: MongoDB Install Procedure

Fuente. - Tomado de <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/#install-mongodb-community-edition>

Paso 04: selecciona la opción “MongoDB Community Center”.



Choose which type of deployment is best for you

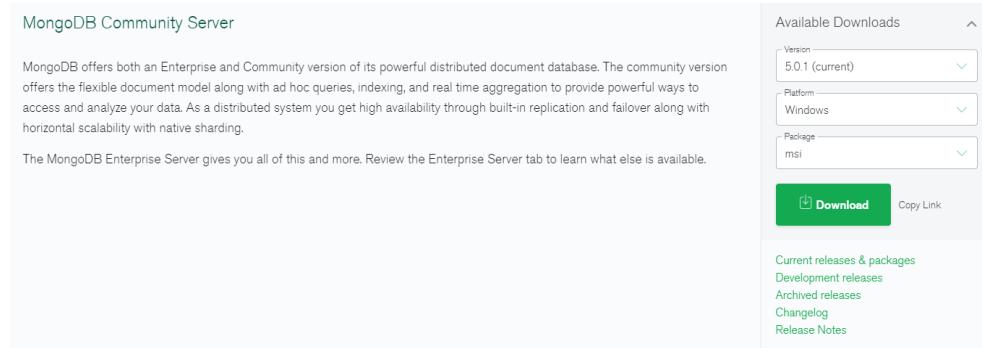
MongoDB Enterprise Server

- MongoDB Community Server (selected)
- MongoDB Ops Manager
- MongoDB Enterprise Kubernetes Operator
- MongoDB Community Kubernetes Operator
- MongoDB Charts

Figura 30: MongoDB Install Procedure

Fuente. - Tomado de <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/#install-mongodb-community-edition>

Paso 05: clic en “download” dejando por defecto la información del formulario.



MongoDB Community Server

MongoDB offers both an Enterprise and Community version of its powerful distributed document database. The community version offers the flexible document model along with ad hoc queries, indexing, and real time aggregation to provide powerful ways to access and analyze your data. As a distributed system you get high availability through built-in replication and failover along with horizontal scalability with native sharding.

The MongoDB Enterprise Server gives you all of this and more. Review the Enterprise Server tab to learn what else is available.

Available Downloads

- Version: 5.0.1 (current)
- Platform: Windows
- Package: msi

Current releases & packages

- Development releases
- Archived releases
- Changelog
- Release Notes

Download Copy Link

Figura 31: MongoDB Install Procedure

Fuente. - Tomado de <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/#install-mongodb-community-edition>

Paso 06: descargar el archivo en carpeta “descarga” de la computadora.

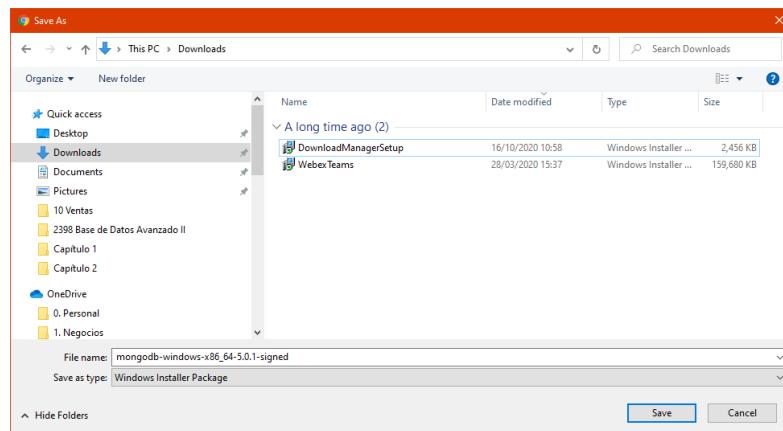


Figura 32: Captura de Pantalla 1
Fuente .- Elaboración Propia

Paso 07: descargado el instalador de MongoDB y hacerle doble clic en **Next**.



Figura 33: Captura de Pantalla 2
Fuente .- Elaboración Propia

Paso 08: aceptar los términos de licencia y hacerle clic en **Next**.

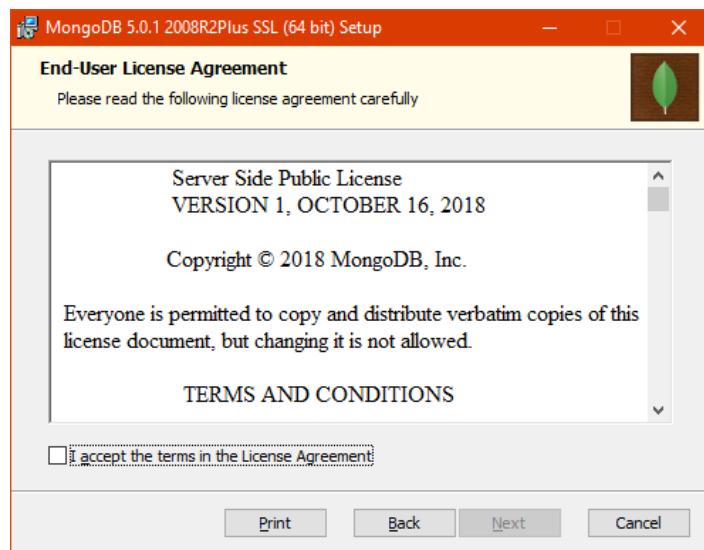


Figura 34: Captura de Pantalla 3
Fuente .- Elaboración Propia

Paso 09: clic en botón **Complete**.

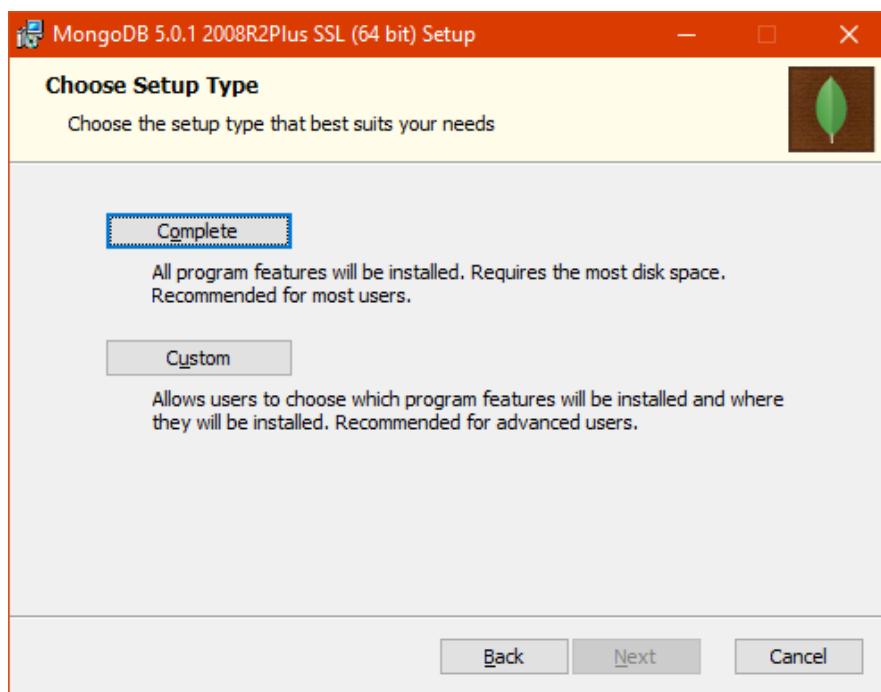


Figura 35: Captura de Pantalla 4
Fuente .- Elaboración Propia

Paso 10: seleccionar la ubicación de instalación.

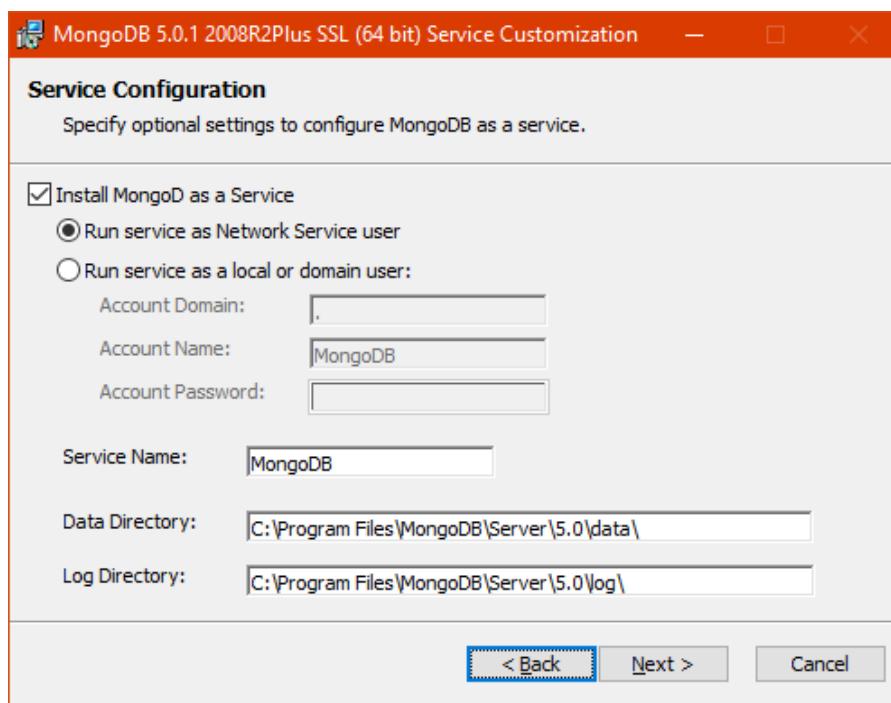


Figura 36: Captura de Pantalla 5

Fuente .- Elaboración Propia

Paso 11: seleccionar “Install MongoDB Compass” y clic en **Next**.



Figura 37: Captura de Pantalla 6

Fuente .- Elaboración Propia

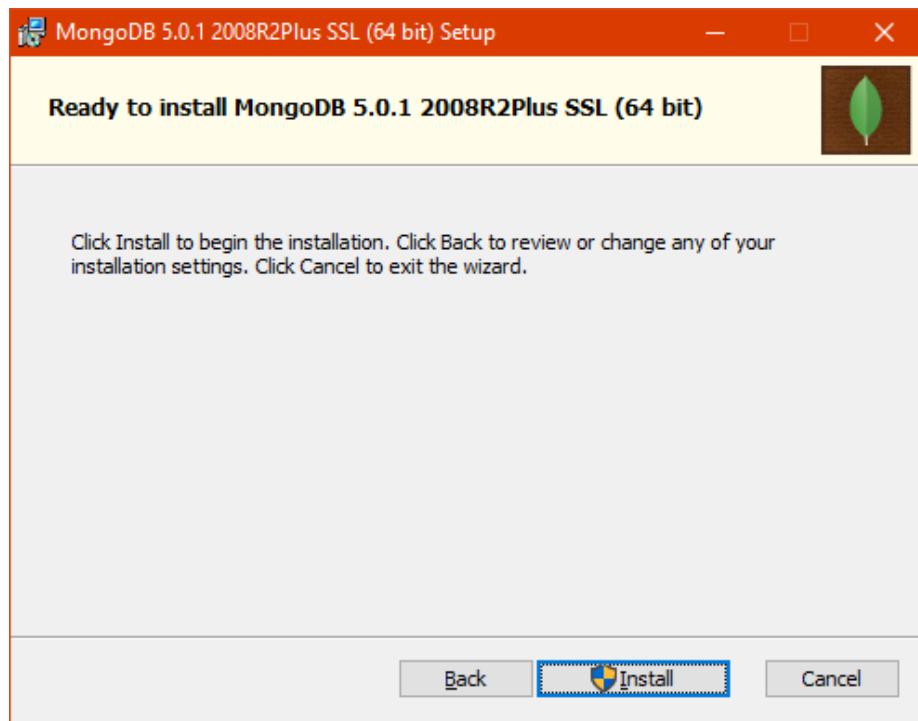
Paso 12: clic en **Install**.

Figura 38: Captura de Pantalla 7

Fuente .- Elaboración Propia

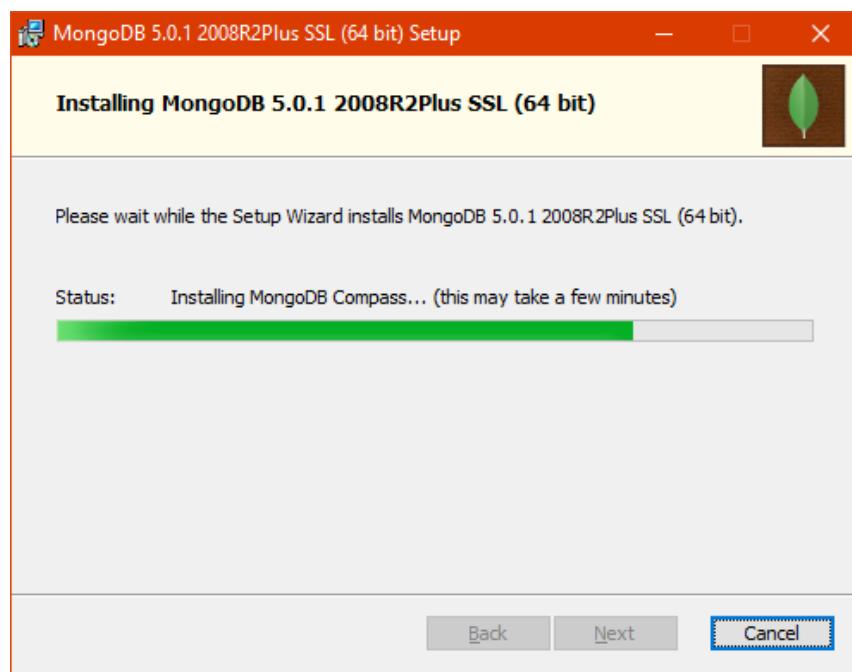
Paso 13: espere mientras el instalador complete el proceso.

Figura 39: Captura de Pantalla 8

Fuente .- Elaboración Propia

2.2.3. Configuración

La ubicación por defecto de los archivos de MongoDB se almacena:



Figura 40: Captura de Pantalla 9

Fuente .- Elaboración Propia

Nota: se recomienda no cambiar la ruta propuesta por el instalador.

En el proceso de instalación, el instalador ofrece la opción de especificar la ubicación de la base de datos y archivos de registro del sistema.

Name	Date modified	Type	Size
bin	3/08/2021 17:11	File folder	
data	3/08/2021 17:50	File folder	
log	3/08/2021 17:11	File folder	
LICENSE-Community	19/07/2021 22:45	Text Document	30 KB
MPL-2	19/07/2021 22:45	File	17 KB
README	19/07/2021 22:45	File	2 KB
THIRD-PARTY-NOTICES	19/07/2021 22:45	File	76 KB

Figura 41: Captura de Pantalla 10

Fuente .- Elaboración Propia

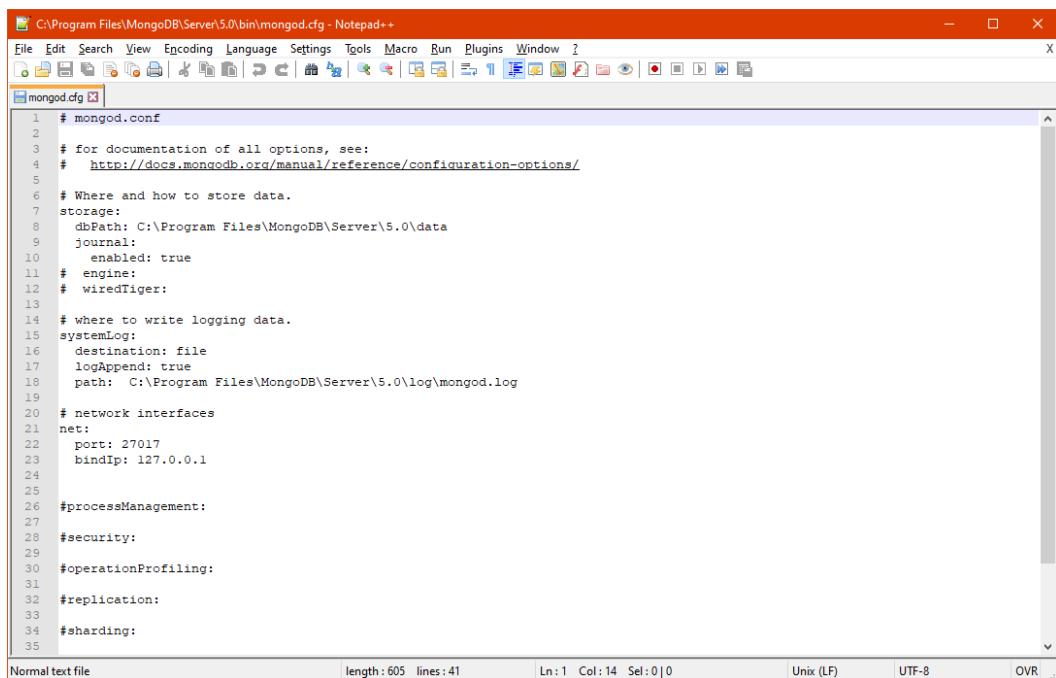
El archivo de configuración **mongo.cfg** contiene las ubicaciones de las bases de datos y los archivos de registro

Name	Date modified	Type	Size
InstallCompass	19/07/2021 22:45	Windows PowerS...	2 KB
mongo	19/07/2021 23:10	Application	21,643 KB
mongod.cfg	3/08/2021 17:14	CFG File	1 KB
mongod	20/07/2021 00:14	Application	45,657 KB
mongod.pdb	20/07/2021 00:14	PDB File	515,844 KB
mongos	19/07/2021 23:35	Application	29,050 KB
mongos.pdb	19/07/2021 23:35	PDB File	304,676 KB

Figura 42: Captura de Pantalla 11

Fuente .- Elaboración Propia

Cabe resaltar, que el instalador de MongoDB crea el archivo de manera automática. A continuación, una vista del contenido con información por default.



```

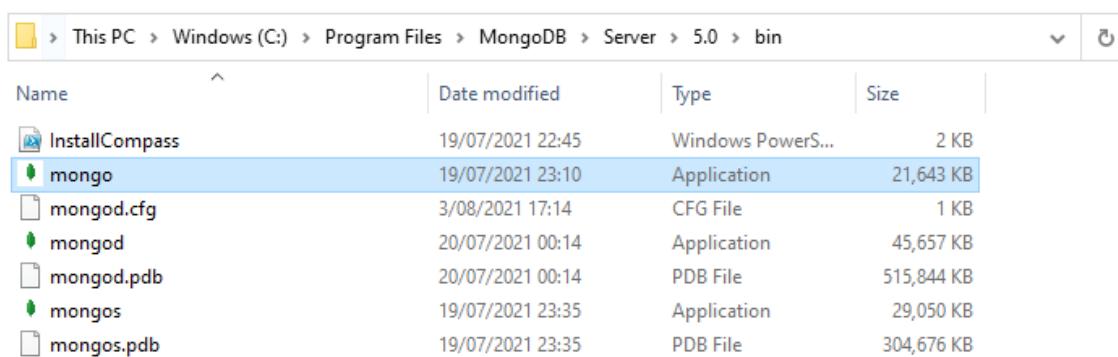
C:\Program Files\MongoDB\Server\5.0\bin\mongod.cfg - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? 
File New Open Save Save As Save All Print Find Replace Go To Preferences Plugins Help 
mongod.cfg x
1 # mongod.conf
2
3 # for documentation of all options, see:
4 #   http://docs.mongodb.org/manual/reference/configuration-options/
5
6 # Where and how to store data.
7 storage:
8   dbPath: C:\Program Files\MongoDB\Server\5.0\data
9   journal:
10    enabled: true
11   # engine:
12   #   wiredTiger:
13
14   # where to write logging data.
15 systemLog:
16   destination: file
17   logAppend: true
18   path: C:\Program Files\MongoDB\Server\5.0\log\mongod.log
19
20 # network interfaces
21 net:
22   port: 27017
23   bindIp: 127.0.0.1
24
25
26 #processManagement:
27
28 #security:
29
30 #operationProfiling:
31
32 #replication:
33
34 #sharding:
35
Normal text file length : 605 lines : 41 Ln : 1 Col : 14 Sel : 0 | 0 Unix (LF) UTF-8 OVR : ...

```

Figura 43: Captura de Pantalla 12

Fuente .- Elaboración Propia

Para iniciar MongoDB debe accederse a la carpeta de instalación, clic en **mongo.exe**.

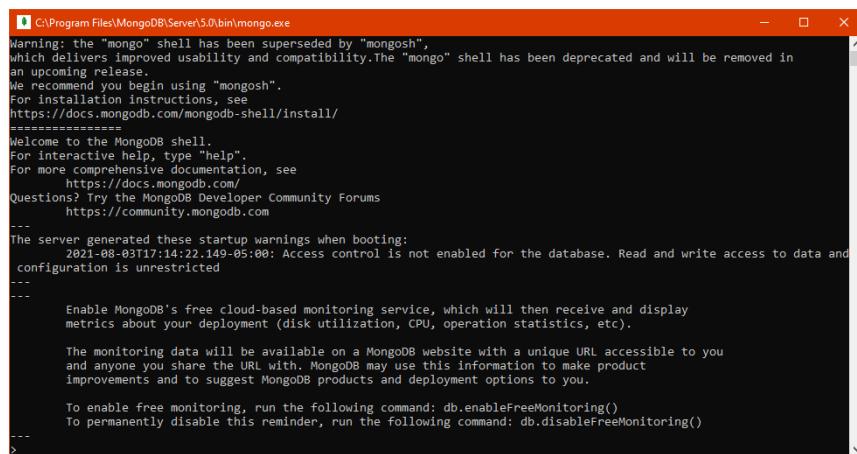


This PC > Windows (C:) > Program Files > MongoDB > Server > 5.0 > bin				
Name	Date modified	Type	Size	
InstallCompass	19/07/2021 22:45	Windows PowerS...	2 KB	
mongo	19/07/2021 23:10	Application	21,643 KB	
mongod.cfg	3/08/2021 17:14	CFG File	1 KB	
mongod	20/07/2021 00:14	Application	45,657 KB	
mongod.pdb	20/07/2021 00:14	PDB File	515,844 KB	
mongos	19/07/2021 23:35	Application	29,050 KB	
mongos.pdb	19/07/2021 23:35	PDB File	304,676 KB	

Figura 44: Captura de Pantalla 13

Fuente .- Elaboración Propia

A continuación, las librerías de MongoDB se cargan, dejándolo activo el sistema.



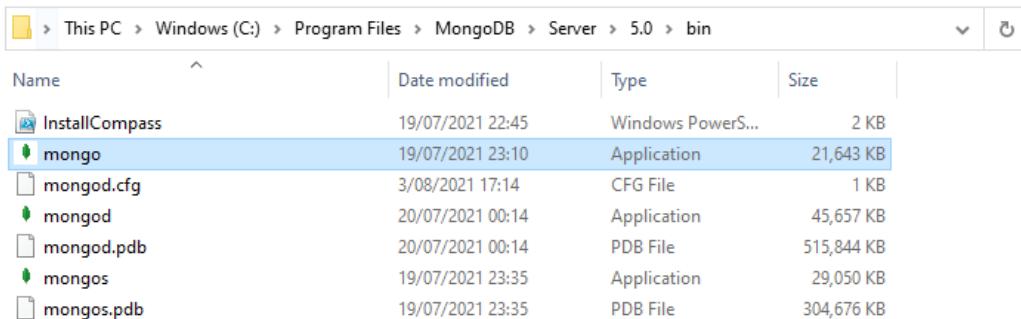
```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility. The "mongo" shell has been deprecated and will be removed in
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
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:
2021-08-03T17:14:22.149-05:00: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted
...
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 45: Captura de Pantalla 14

Fuente .- Elaboración Propia

2.2.4. Consola de trabajo

Recordemos que MongoDB es una base de datos NoSQL y no se debe de modelar los datos en este tipo de base de datos como en las bases de datos relacionales, por ello el modelo de datos es DESNORMALIZADO. Esto significa cambiar totalmente la forma en que hemos gestionado las bases de datos en cursos de ciclos anteriores. Dicho esto, no es que un esquema sea mejor que el otro, ambos son válidos, pero se acomoda mejor según el contexto donde se implementa. (Saiz, 2014).

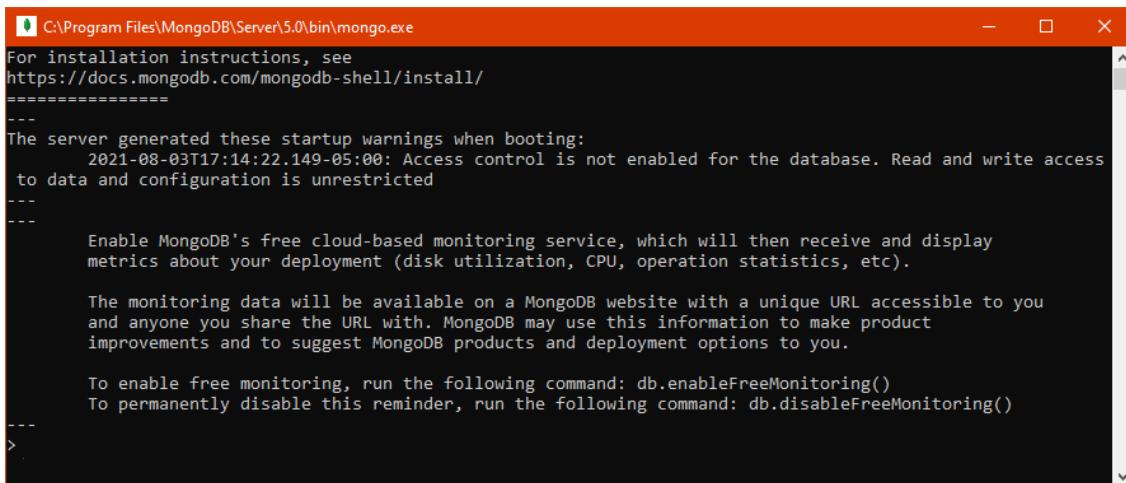


Name	Date modified	Type	Size
InstallCompass	19/07/2021 22:45	Windows PowerS...	2 KB
mongo	19/07/2021 23:10	Application	21,643 KB
mongod.cfg	3/08/2021 17:14	CFG File	1 KB
mongod	20/07/2021 00:14	Application	45,657 KB
mongod.pdb	20/07/2021 00:14	PDB File	515,844 KB
mongos	19/07/2021 23:35	Application	29,050 KB
mongos.pdb	19/07/2021 23:35	PDB File	304,676 KB

Figura 46: Captura de Pantalla 15

Fuente .- Elaboración Propia

Dicho esto, ir a la carpeta de la instalación de MongoDB y hacer clic a **mongodb.exe**.



```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
=====
---+
The server generated these startup warnings when booting:
    2021-08-03T17:14:22.149-05:00: Access control is not enabled for the database. Read and write access
to data and configuration is unrestricted
---+
---+
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

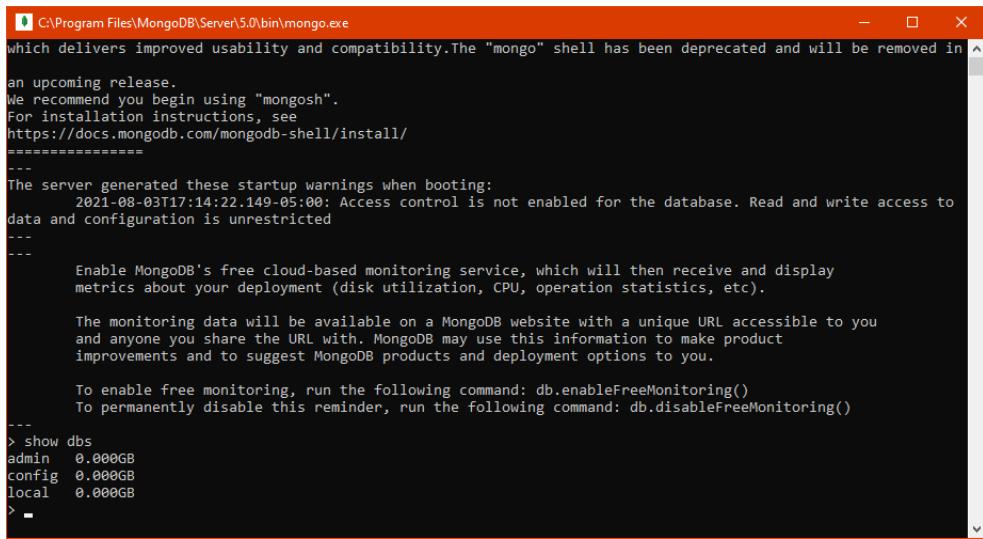
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---+
> .
```

Figura 47: Captura de Pantalla 16
Fuente .- Elaboración Propia

A continuación, describiremos algunos comandos básicos de uso del MongoDB.

- **show dbs:** muestra las bases de datos NoSQL del servidor.



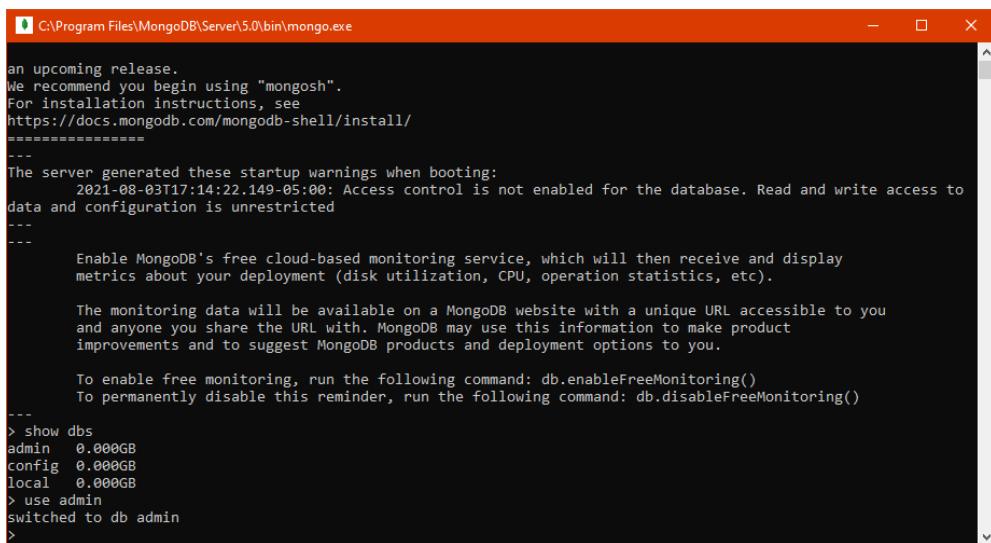
```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
which delivers improved usability and compatibility. The "mongo" shell has been deprecated and will be removed in a
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
=====
---+
The server generated these startup warnings when booting:
    2021-08-03T17:14:22.149-05:00: Access control is not enabled for the database. Read and write access to
data and configuration is unrestricted
---+
---+
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()
---+
> show dbs
admin  0.000GB
config  0.000GB
local   0.000GB
> .
```

Figura 48: Captura de Pantalla 17
Fuente .- Elaboración Propia

- **use [nombre base de datos] dbs:** hace uso de la base de datos existente o la crea y se mostrará listada con show dbs.



```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe

an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
=====
---
The server generated these startup warnings when booting:
    2021-08-03T17:14:22.149-05:00: Access control is not enabled for the database. Read and write access to
data and configuration is unrestricted
---
---
    Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

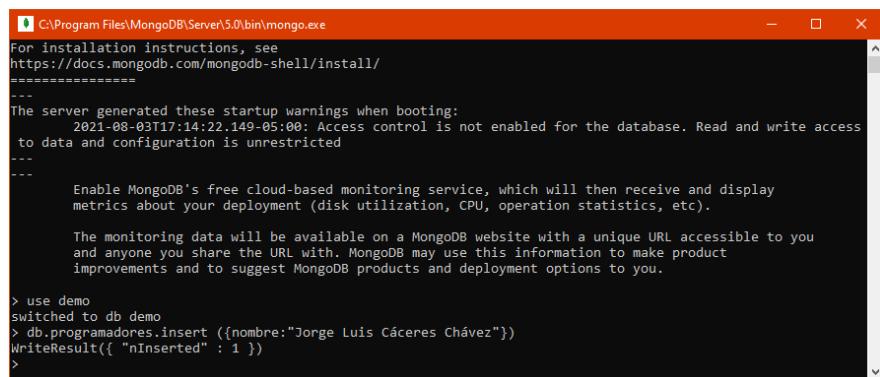
    The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

    To enable free monitoring, run the following command: db.enableFreeMonitoring()
    To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
> use admin
switched to db admin
>
```

Figura 49: Captura de Pantalla 18

Fuente .- Elaboración Propia

- **db.[colección].insert([documento en formato JSON]):** inserta y crea una colección de datos en formato JSON. Las colecciones se crean automáticamente en MongoDB, una vez que insertas un elemento o colección.



```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe

For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
=====
---
The server generated these startup warnings when booting:
    2021-08-03T17:14:22.149-05:00: Access control is not enabled for the database. Read and write access to
data and configuration is unrestricted
---
---
    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.

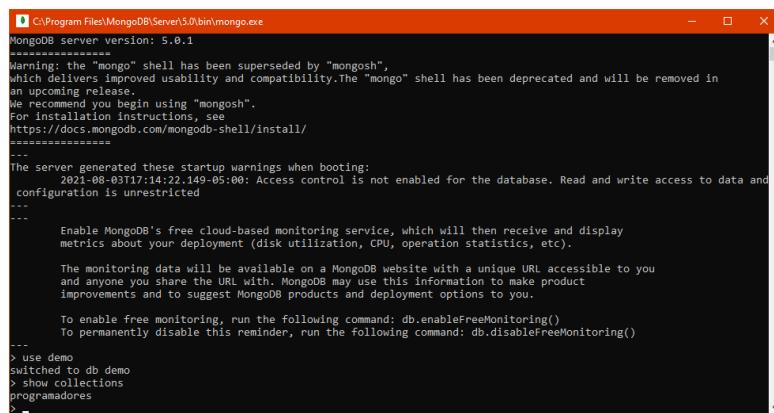
> use demo
switched to db demo
> db.programadores.insert ({nombre:"Jorge Luis Cáceres Chávez"})
WriteResult({ "nInserted" : 1 })
>
```

Figura 50: Captura de Pantalla 19

Fuente .- Elaboración Propia

Nota: las colecciones equivalen a tablas en las bases de datos relacionales

- **show collections:** muestra las colecciones de la base datos activa, recordar primero seleccionar la base de datos con el comando **use [nombre de base de datos]**.

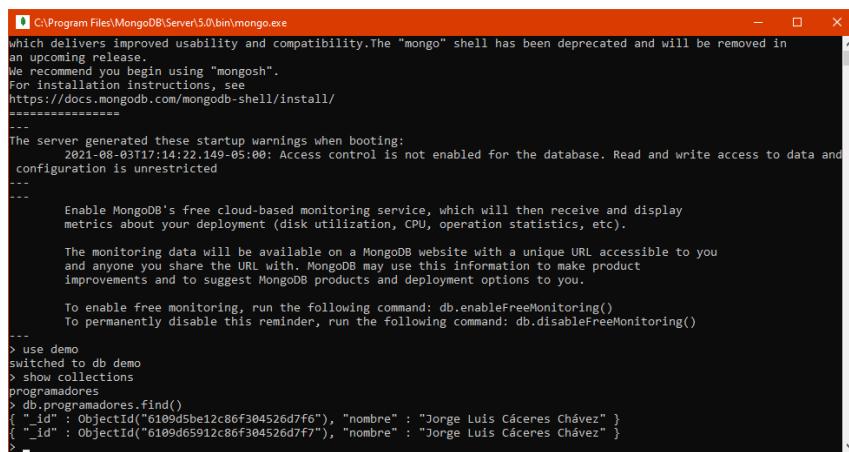


```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
MongoDB server version: 5.0.1
=====
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
=====
The server generated these startup warnings when booting:
2021-08-03T17:14:22.149+05:00: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted
=====
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()
>
> use demo
switched to db demo
> show collections
programadores
```

Figura 51: Captura de Pantalla 20

Fuente.- Elaboración Propia

- **db.[colección].find():** muestra la lista de documentos (“registros”) de una colección.

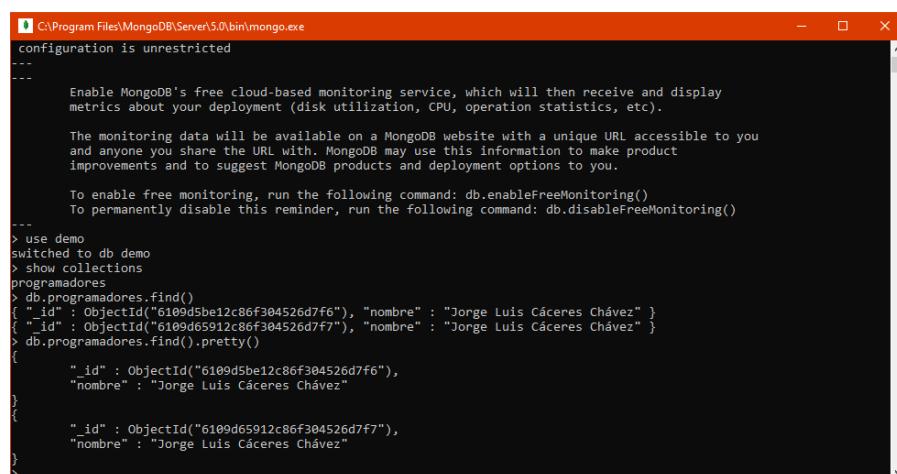


```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
=====
The server generated these startup warnings when booting:
2021-08-03T17:14:22.149+05:00: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted
=====
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()
>
> use demo
switched to db demo
> show collections
programadores
> db.programadores.find()
{
  "_id" : ObjectId("6109d5be12c86f304526d7f6"), "nombre" : "Jorge Luis Cáceres Chávez"
}
{
  "_id" : ObjectId("6109d65912c86f304526d7f7"), "nombre" : "Jorge Luis Cáceres Chávez"
}
>
```

Figura 52: Captura de Pantalla 21

Fuente.- Elaboración Propia

- **pretty():** mejora la salida de resultados de la consola.



```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
configuration is unrestricted
=====
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()
>
> use demo
switched to db demo
> show collections
programadores
> db.programadores.find()
{
  "_id" : ObjectId("6109d5be12c86f304526d7f6"), "nombre" : "Jorge Luis Cáceres Chávez"
}
{
  "_id" : ObjectId("6109d65912c86f304526d7f7"), "nombre" : "Jorge Luis Cáceres Chávez"
}
> db.programadores.find().pretty()
{
  "_id" : ObjectId("6109d5be12c86f304526d7f6"),
  "nombre" : "Jorge Luis Cáceres Chávez"
}
{
  "_id" : ObjectId("6109d65912c86f304526d7f7"),
  "nombre" : "Jorge Luis Cáceres Chávez"
}
>
```

Figura 53: Captura de Pantalla 22

Fuente.- Elaboración Propia

2.2.5. Tipo de datos

Los valores en MongoDB no son solo simples Strings. MongoDB tiene varios tipos de datos permitidos en los valores de sus documentos.

- **String:** tipo de dato texto debe ser válida para UTF-8.
- **Integer:** tipo de dato numérico puede ser de 32 o 64 bits.
- **Boolean:** tipo de dato lógico (verdadero / falso).
- **Double:** tipo de dato para almacenar valores de punto flotante.
- **Arrays:** tipo de dato arreglo o múltiples valores en una clave.
- **Timestamp:** tipo de dato de tiempo. (CodeHoven, 2019).
- **Date:** tipo de dato de fecha y hora.
- **Object:** tipo de dato para documentos incrustados.
- **ObjectId:** tipo de dato para almacenar la ID del documento.
- **Binary:** tipo de dato para almacenar binarios.

Resumen

1. MongoDB es una base de datos documental de código abierto que ofrece alto rendimiento, alta disponibilidad y escalado automático.
2. El mayor uso de MongoDB es en proyectos de Big Data con grandes usuarios y complejidad en la gestión de la información.
3. Cada registro o conjunto de datos se denomina documento y pueden agruparse en colecciones.

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://www.youtube.com/watch?v=2GU938abfQg>
- <https://www.youtube.com/watch?v=CuAYLX6reXE&t=21s>
- <https://www.youtube.com/watch?v=5xxUhaWrl4U>



CRUD

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno interactúa con la base de datos MongoDB.

TEMARIO

3.1 Tema 7 : ¿Qué es CRUD?

- 3.1.1 : CREATE
- 3.1.2 : READ
- 3.1.3 : UPDATE
- 3.1.4 : DELETE

ACTIVIDADES PROPUESTAS

- Los alumnos realizan las lecturas propuestas.
- Los alumnos aplican los métodos principales CRUD.

3.1. ¿QUÉ ES CRUD?

El término **CRUD** está vinculado a la gestión de los datos. **CRUD** hace referencia a un acrónimo en el que se reúnen las primeras letras de las 4 operaciones fundamentales de las aplicaciones persistentes en sistemas de bases de datos.

- Create (Crear registros)
- Read (Leer registros)
- Update (Actualizar registros)
- Delete (Borrar registros)

En pocas palabras, **CRUD** resume las funciones requeridas por un usuario para crear y gestionar datos. Varios procesos de gestión de datos están basados en **CRUD**, en los que dichas operaciones están específicamente adaptadas a los requisitos del sistema y de usuario, ya sea para la gestión de bases de datos o para el uso de aplicaciones.



Figura 54: CRUD

Fuente. - Tomado de <https://webservice.com.ec/blog/que-es-crud/>

En MongoDB los nombres están asociados a las funciones siguientes:

Tabla 2
MongoDB

Acción	Verbo
CREATE	Insert
READ	Find
UPDATE	Update
DELETE	Remove

Nota. Elaboración Propia

Métodos para insertar documentos en una colección.

- db.<collection>.insertOne({JSON-Document})
- db.<collection>.insertMany([{JSON-Document},{Other-JSON-Document}, {...}])

Métodos para leer documentos de una colección.

- db.<collection>.find()
- db.<collection>.find({"clave": "valor"}, {"clave": valor})

Métodos para actualización de documentos de una colección.

- db.<collection>.updateOne({filtro}, {"clave": "valor"})
- db.<collection>.updateMany({filtro}, {"clave": "valor"})
- db.<collection>.replaceOne({filtro}, {"clave": "valor"})

Métodos para eliminación de documentos de una colección.

- db.<collection>.deleteOne({"filter"})
- db.<collection>.deleteMany({"filter"})

(Tecnologias-Informacion.com, 2018).

Nota: un documento MongoDB, equivale a una tabla en una base de datos relacional.

3.1.1. CREATE

La creación de documentos en MongoDB es fácil, recuerda que en estas bases de datos los documentos están en formato JSON y almacenados en formato BSON que hemos comentado en capítulos anteriores. Teniendo en cuenta esto, todo lo que necesitas trabajar en se tiene que hacer solo y exclusivamente en formato JSON. Esta condición es vital, y a pesar de que puede ser restrictivo inicialmente, es muy importante ya que permite la integración para intercambios de información con otras soluciones.

Considerar las siguientes restricciones y limitaciones.

- Si el campo “_id” este duplicado se lanza una excepción
- El tamaño máximo de un documento BSON es de 16 MB
- El campo “_id” es usado como clave primaria
- Ningún campo puede comenzar con \$ y tampoco puede con un “.”

Ahora bien, el campo “_id” es un campo **OBLIGATORIO**, este campo se puede asignar, pero ¿qué sucede si no especificamos el campo “_id”? . MongoDB le genera un valor de 12 bytes de tipo ObjectId donde:

- 4 bytes son de timestamp (fecha y hora).
- 3 bytes del identificador de la computadora.
- 2 bytes del número de proceso.
- 3 bytes de un contador que comienza con un número aleatorio.

A continuación, detallamos los comandos.

- insertOne(): permite insertar un documento dentro de una colección.

- Ejemplo por consola asignando el “_id”.

```
⚡ C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeRepartidor.insertOne({
... _id: "R001",
... apPaterno: "PATIÑO",
... apMaterno: "VARGAS",
... nombres: "JACINTO LUIS",
... direccion: {domicilio: "AV. TACNA 482", departamento: "LIMA",
... provincia: "LIMA", distrito: "SAN MARTIN DE PORRES"}
... }
...
{
  "acknowledged" : true, "insertedId" : "R001"
}
>
>
```

Figura 55: Captura de Pantalla 23
Fuente .- Elaboración Propia

- Resultado de la operación.

```
⚡ C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeRepartidor.find().pretty()
{
  "_id" : "R001",
  "apPaterno" : "PATIÑO",
  "apMaterno" : "VARGAS",
  "nombres" : "JACINTO LUIS",
  "direccion" : {
    "domicilio" : "AV. TACNA 482",
    "departamento" : "LIMA",
    "provincia" : "LIMA",
    "distrito" : "SAN MARTIN DE PORRES"
  }
}
>
```

Figura 56: Captura de Pantalla 24
Fuente .- Elaboración Propia

- Ejemplo por MongoDB Compass asignando el “_id”.

Insert Document

```

1   _id : "R002"                               String
2   apPaterno : "PATIÑO"                      String
3   apMaterno : "VARGAS"                       String
4   nombres : "JACINTO LUIS"                   String
5   ✓ dirección : Object                        Object
6     domicilio : "AV. TEJADA 150"            String
7     departamento : "LIMA"                   String
8     provincia : "LIMA"                      String
9     distrito : "PACHACAMAC"                 String

```

CANCEL **INSERT**

Figura 57: Captura de Pantalla 25
Fuente .- Elaboración Propia

- Resultado de la operación.

```

> _id: "R002"
  apPaterno: "PATIÑO"
  apMaterno: "VARGAS"
  nombres: "JACINTO LUIS"
  ✓ dirección: Object
    domicilio: "AV. TEJADA 150"
    departamento: "LIMA"
    provincia: "LIMA"
    distrito: "PACHACAMAC"

```

Figura 58: Captura de Pantalla 26
Fuente .- Elaboración Propia

- Ejemplo por MongoDB Compass sin incluir el “_id”.

```

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeHelados.insertOne({
...   descripcion: "Maracuyá al agua",
...   composicion: "Helado de agua sabor maracuyá, sembrado con variegato de maracuyá.",
...   stockActual: 100,
...   precio: 15.54
... })
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5da14f13a51dcef20f4a7573")
}

```

Figura 59: Captura de Pantalla 27
Fuente .- Elaboración Propia

- Resultado de la operación por consola.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeHelados.find().pretty()
{
    "_id" : ObjectId("5da14f13a51dcef20f4a7573"),
    "descripcion" : "Maracuyá al agua",
    "composicion" : "Helado de agua sabor maracuyá, sembrado con variegato de maracuyá.",
    "stockActual" : 100,
    "precio" : 15.54
}
> ■
```

Figura 60: Captura de Pantalla 28

Fuente .- Elaboración Propia

- Resultado de la operación por consola.

Insert Document

1	_id : ObjectId("5da14f9fd3b45417fc5b149e")	ObjectId
2	descripcion : "Naranja"	String
3	composicion : "Helado de agua de naranja con jugo de naranja"	String
4	stockActual : 95	Int32
5	precio : 16.54	Double

Figura 61: Captura de Pantalla 29

Fuente .- Elaboración Propia

Nota: la propiedad `_id` fue ingresada por el MongoDB de manera automática. Es parecido al Identity de las bases de datos relacionales, sin embargo, aquí es una cadena con un valor que incluye un algoritmo que hace imposible su repetición y no tiene límites de tamaño del INT.

- **insertMany():** permite insertar varios documentos dentro de una colección.
- Ejemplo por consola sin asignar el “_id” para insertar 3 documentos.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeRepartidor.insertMany([
... {apPaterno: "PATIÑO", apMaterno: "VARGAS", nombres: "JACINTO",
... domicilio: "AV. TACNA 482 - CERCADO DE LIMA"},
... {apPaterno: "PEZO", apMaterno: "CHOQUEHUANCA", nombres: "FILIBERTO",
... domicilio: "AV. TEJADA 150 - SAN ISIDRO"},
... {apPaterno: "RAMIREZ", apMaterno: "TAVARA", nombres: "JOSE LUIS",
... domicilio: "AV. JOSE PARDO 620 - MIRAFLORES"}
... ])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5da1548ca51dcef20f4a7574"),
    ObjectId("5da1548ca51dcef20f4a7575"),
    ObjectId("5da1548ca51dcef20f4a7576")
  ]
}
```

Figura 62: Captura de Pantalla 30
Fuente .- Elaboración Propia

- Resultado de la operación.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeRepartidor.find().pretty()
{
  "_id" : ObjectId("5da1548ca51dcef20f4a7574"),
  "apPaterno" : "PATIÑO",
  "apMaterno" : "VARGAS",
  "nombres" : "JACINTO",
  "domicilio" : "AV. TACNA 482 - CERCADO DE LIMA"
}
{
  "_id" : ObjectId("5da1548ca51dcef20f4a7575"),
  "apPaterno" : "PEZO",
  "apMaterno" : "CHOQUEHUANCA",
  "nombres" : "FILIBERTO",
  "domicilio" : "AV. TEJADA 150 - SAN ISIDRO"
}
{
  "_id" : ObjectId("5da1548ca51dcef20f4a7576"),
  "apPaterno" : "RAMIREZ",
  "apMaterno" : "TAVARA",
  "nombres" : "JOSE LUIS",
  "domicilio" : "AV. JOSE PARDO 620 - MIRAFLORES"
}
```

Figura 63: Captura de Pantalla 31
Fuente .- Elaboración Propia

- Ejemplo de inserción de un documento predefinido.

```
↳ C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> repartidor1 = {
... apPaterno: "RODRIGUEZ", apMaterno: "MACASI", nombres: "JULIO CESAR",
... domicilio: "PASAJE CABO ALARCON 140 - SURQUILLO"
...
{
    "apPaterno" : "RODRIGUEZ",
    "apMaterno" : "MACASI",
    "nombres" : "JULIO CESAR",
    "domicilio" : "PASAJE CABO ALARCON 140 - SURQUILLO"
}
> db.maeRepartidor.insertOne(repartidor1)
{
    "acknowledged" : true,
    "insertedId" : ObjectId("5da155b4a51dcef20f4a7577")
}
```

Figura 64: Captura de Pantalla 32
Fuente .- Elaboración Propia

- Ejemplo de inserción de un array de documentos.

```
↳ C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> repartidor2 = {
... apPaterno: "CERRO", apMaterno: "OLIVERA", nombres: "JESSICA JULIANA",
... domicilio: "JR. MIGUEL ALJOVIN 231 - SAN MIGUEL"
{
    "apPaterno" : "CERRO",
    "apMaterno" : "OLIVERA",
    "nombres" : "JESSICA JULIANA",
    "domicilio" : "JR. MIGUEL ALJOVIN 231 - SAN MIGUEL"
}
> repartidor3 = {
... apPaterno: "MALDONADO", apMaterno: "VASQUEZ", nombres: "ROSA ENITH",
... domicilio: "JR. ANTONIO MIROQUEZADA 376 - CERCADO DE LIMA"
{
    "apPaterno" : "MALDONADO",
    "apMaterno" : "VASQUEZ",
    "nombres" : "ROSA ENITH",
    "domicilio" : "JR. ANTONIO MIROQUEZADA 376 - CERCADO DE LIMA"
}
> db.maeRepartidor.insertMany([repartidor2, repartidor3])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5da1569ba51dcef20f4a7578"),
        ObjectId("5da1569ba51dcef20f4a7579")
    ]
}
>
```

Figura 65: Captura de Pantalla 33
Fuente .- Elaboración Propia

- Ejemplo de inserción desde un archivo JSON.

Paso 1: generar el archivo JSON

```

maeRepartidor.json
1 [ 
2   { "apPaterno": "MUÑOZ", "apMaterno": "PINEDO", "nombres": "MARTIN",
3     "domicilio": "JR. APURIMAC 337 - CERCADO DE LIMA"),
4   { "apPaterno": "URBINA", "apMaterno": "PALACIOS", "nombres": "FELIX RICARDO",
5     "domicilio": "CALLE JAIME HERRERA 170 - SAN MIGUEL"),
6   { "apPaterno": "GARCIA", "apMaterno": "CONTRERAS", "nombres": "DANIEL JAIME",
7     "domicilio": "PROLONGACION LA PUENTE Y CORTEZ 312-316 - SAN MIGUEL"),
8   { "apPaterno": "UBILLUS", "apMaterno": "CASTILLO", "nombres": "NILTON CESAR",
9     "domicilio": "AV .LAS CAMELIAS 710 - SAN ISIDRO"),
10  { "apPaterno": "URBINA", "apMaterno": "CALDERON", "nombres": "RODOLFO",
11    "domicilio": "JR. FUNO 158 - CERCADO DE LIMA")
12 ]
13 ]

```

Figura 66: Captura de Pantalla 34

Fuente .- Elaboración Propia

Paso 2: desde CMD de Windows, ejecuta **mongoimport.exe** con los parámetros que se muestran a continuación.

```

C:\Program Files\MongoDB\Server\4.2\bin>mongoimport.exe --host localhost --port 27017 --db dbHeladeria
--collection maeRepartidor --file D:\repartidor.json --jsonArray
2019-10-12T00:16:08.175-0500      connected to: mongodb://localhost:27017/
2019-10-12T00:16:08.206-0500      5 document(s) imported successfully. 0 document(s) failed to import.
C:\Program Files\MongoDB\Server\4.2\bin>

```

Figura 67: Captura de Pantalla 35

Fuente .- Elaboración Propia

Nota: se asume que el archivo **repartidor.json** se ubica en la unidad d:\

3.1.2. READ

Para obtener los documentos almacenados en una colección MongoDB se debe utilizar el método `find()`. (CodeHoven, 2018).

Sintaxis:

```
db.<collection>.find(filter, projection)
```

Donde:

- Filter: es un filtro por donde pasa los documentos de la colección.
- Projection: muestra u oculta la clave-valor del documento.

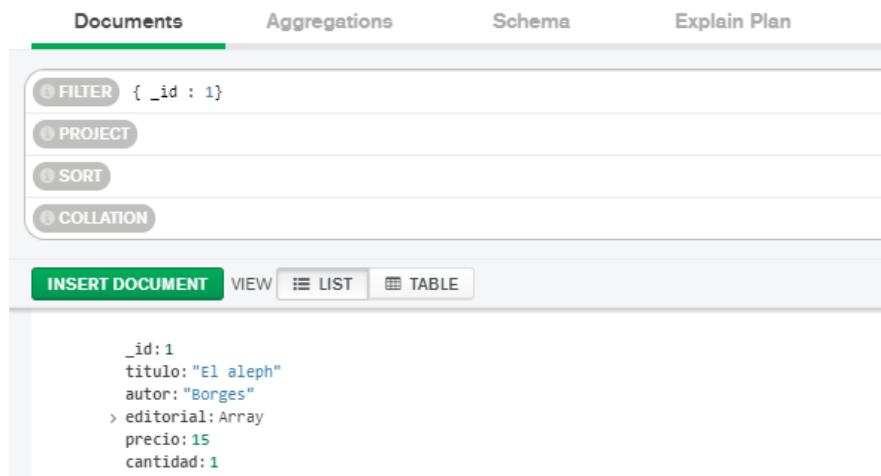
- Ejemplo de búsqueda de un documento con valor `_id = 1`.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
db.maeLibros.find({_id : 1}).pretty()
{
  "_id" : 1,
  "titulo" : "El aleph",
  "autor" : "Borges",
  "editorial" : [
    "Siglo XXI",
    "Planeta"
  ],
  "precio" : 15,
  "cantidad" : 1
}
```

Figura 68: Captura de Pantalla 36
Fuente .- Elaboración Propia

- Ejemplo por MongoDB Compass asignando el “_id = 1”.



The screenshot shows the MongoDB Compass interface with the 'Documents' tab selected. At the top, there are four buttons: FILTER, PROJECT, SORT, and COLLATION. Below them is a green 'INSERT DOCUMENT' button and a 'VIEW' dropdown with 'LIST' and 'TABLE' options. The main area displays the following document:

```
_id: 1
titulo: "El aleph"
autor: "Borges"
editorial: Array
precio: 15
cantidad: 1
```

Figura 69: Captura de Pantalla 37
Fuente .- Elaboración Propia

- Ejemplo de búsqueda de libros por consola con cantidad = 100.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.find({cantidad : 100}).pretty()
{
  "_id" : 4,
  "titulo" : "Java en 10 minutos",
  "autor" : "Mario Molina",
  "editorial" : [
    "Siglo XXI"
  ],
  "precio" : 45,
  "cantidad" : 100
}
{
  "_id" : 3,
  "titulo" : "Aprenda PHP",
  "autor" : "Mario Molina",
  "editorial" : [
    "Siglo XXI",
    "Planeta"
  ],
  "precio" : 50,
  "cantidad" : 100
}
```

Figura 70: Captura de Pantalla 38
Fuente .- Elaboración Propia

- Ejemplo de búsqueda de libros por MongoDB Compass con cantidad = 100.

The screenshot shows the MongoDB Compass interface with the 'Documents' tab selected. A search filter is applied: {cantidad : 100}. The results show two documents:

- Document 1: _id: 4, titulo: "Java en 10 minutos", editorial: Array, precio: 45, cantidad: 100
- Document 2: _id: 3, titulo: "Aprenda PHP", autor: "Mario Molina", editorial: Array, precio: 50, cantidad: 100

Figura 71: Captura de Pantalla 39

Fuente .- Elaboración Propia

- Ejemplo de búsqueda por consola de varios campos a la vez.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.find({precio: 50, cantidad: 12}).pretty()
{
  "_id" : 2,
  "titulo" : "Martin Fierro",
  "autor" : "Jose Hernandez",
  "editorial" : [
    "Siglo XXI"
  ],
  "precio" : 50,
  "cantidad" : 12
}
>
```

Figura 72: Captura de Pantalla 40

Fuente .- Elaboración Propia

Nota: el comando **pretty()** permite ordenar los datos de salida por consulta.

- Ejemplo de búsqueda por MongoDB Compass de varios campos a la vez.

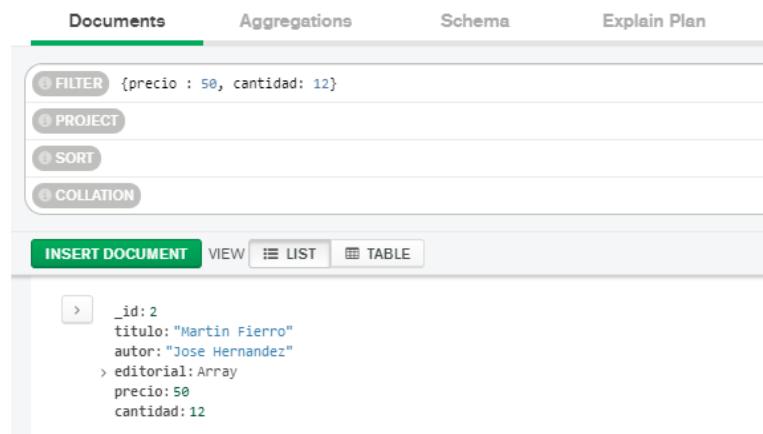


Figura 73: Captura de Pantalla 41

Fuente .- Elaboración Propia

3.1.3. UPDATE

Permite actualizar los documentos almacenados de una colección.

Sintaxis:

```
db.<collection>.updateOne(query, update);
db.<collection>.updateMany(query, update);
db.<collection>.replaceOne(query, update);
```

Donde:

- Query: elementos que se van a seleccionar.
- Update: cambios que se van a realizar.

A continuación, detallamos los comandos.

- **updateOne ()**: permite actualizar un documento dentro de una colección.

- Ejemplo de evidencia por consola del documento a ser actualizado.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.find({"_id": 1}).pretty()
{
    "_id" : 1,
    "título" : "El aleph",
    "autor" : "Borges",
    "editorial" : [
        "Siglo XXI",
        "Planeta"
    ],
    "precio" : 20,
    "cantidad" : 50
}
```

Figura 74: Captura de Pantalla 42

Fuente .- Elaboración Propia

Nota: el comando **pretty()** permite ordenar los datos de salida por consulta.

- Ejemplo por consola antes de actualizar un documento.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.updateOne({ _id : {$eq:1}}, {$set: {precio:15, cantidad:1}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Figura 75: Captura de Pantalla 43
Fuente .- Elaboración Propia

Donde:

- El primer parámetro indica el documento a modificar.
- El segundo parámetro los campos y valores a modificarse con operador \$set.

En bases de datos NoSQL los documentos pueden tener distintas cantidades de campos. Por ejemplo, si quieres adicionar el campo **Descripción** al libro con '_id' 4 debes utilizar la siguiente sintaxis.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.updateOne({ _id : {$eq:4}}, {$set: {descripcion: "Cada unidad
trata un tema fundamental de Java desde 0"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

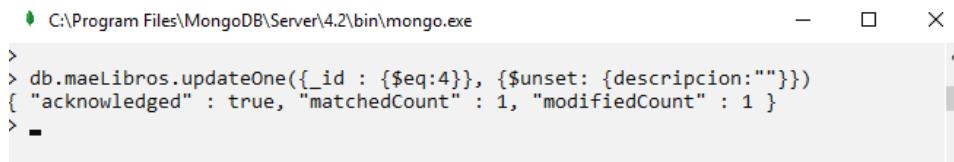
Figura 76: Captura de Pantalla 44
Fuente .- Elaboración Propia

Luego de ejecutar el comando, el documento “_id 4” contiene un nuevo campo llamado **DESCRIPCION** con el valor “Cada unidad trata un tema fundamental de Java desde 0.”.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.find({_id:4}).pretty()
{
    "_id" : 4,
    "titulo" : "Java en 10 minutos",
    "editorial" : [
        "Siglo XXI"
    ],
    "precio" : 45,
    "cantidad" : 1,
    "descripcion" : "Cada unidad trata un tema fundamental de Java desde 0"
}
>
```

Figura 77: Captura de Pantalla 45
Fuente .- Elaboración Propia

También es posible eliminar un campo de un documento utilizando el operador **\$unset**. Por ejemplo: podemos eliminar el campo creado en el documento `_id` 4.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.updateOne({_id : {$eq:4}}, {$unset: {descripcion:""}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> -
```

Figura 78: Captura de Pantalla 46
Fuente .- Elaboración Propia

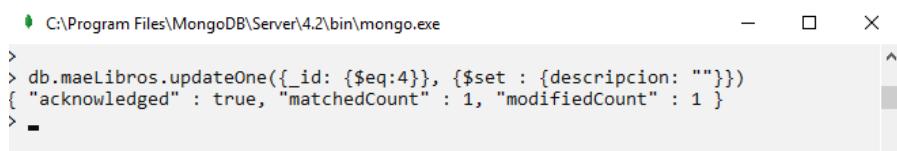
- Evidencia por consola del documento actualizado.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.find({_id:4}).pretty()
{
    "_id" : 4,
    "titulo" : "Java en 10 minutos",
    "editorial" : [
        "Siglo XXI"
    ],
    "precio" : 45,
    "cantidad" : 1
}
> -
```

Figura 79: Captura de Pantalla 47
Fuente .- Elaboración Propia

Es importante entender que mediante el operador **\$unset** se elimina el campo y con el operador **\$set** modifacas el contenido del campo.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.updateOne({_id: {$eq:4}}, {$set : {descripcion: ""}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> -
```

Figura 80: Captura de Pantalla 48
Fuente .- Elaboración Propia

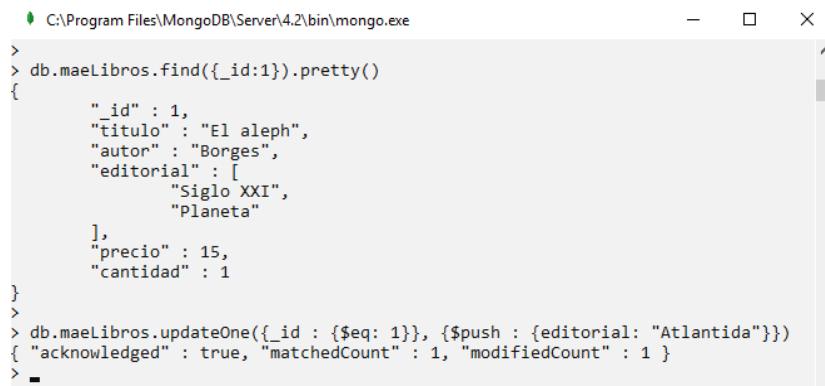
El campo **Descripción** sigue existiendo luego del comando, pero almacena texto vacío y si no existía el campo recuerda que MongoDB lo crea.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.find({_id:4}).pretty()
{
    "_id" : 4,
    "titulo" : "Java en 10 minutos",
    "editorial" : [
        "Siglo XXI"
    ],
    "precio" : 45,
    "cantidad" : 1,
    "descripcion" : ""
}
> -
```

Figura 81: Captura de Pantalla 49
Fuente .- Elaboración Propia

Además, se dispone de operadores de modificación para arreglos.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.find({_id:1}).pretty()
{
    "_id" : 1,
    "titulo" : "El aleph",
    "autor" : "Borges",
    "editorial" : [
        "Siglo XXI",
        "Planeta"
    ],
    "precio" : 15,
    "cantidad" : 1
}
>
> db.maeLibros.updateOne({_id : {$eq: 1}}, {$push : {editorial: "Atlantida"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Figura 82: Captura de Pantalla 50

Fuente .- Elaboración Propia

Luego de ejecutarse el método **updateOne()**, el arreglo 'editorial' tiene un nuevo componente para el documento con _id 1.

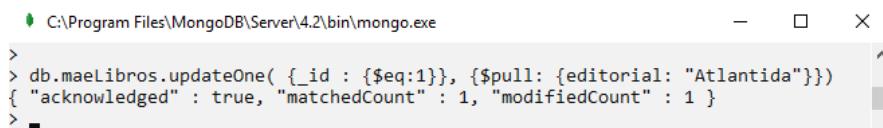


```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.find({_id:1}).pretty()
{
    "_id" : 1,
    "titulo" : "El aleph",
    "autor" : "Borges",
    "editorial" : [
        "Siglo XXI",
        "Planeta",
        "Atlantida"
    ],
    "precio" : 15,
    "cantidad" : 1
}
>
```

Figura 83: Captura de Pantalla 51

Fuente .- Elaboración Propia

De forma similar para eliminar un elemento del arreglo utilizamos el operador **\$pull**.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.updateOne( {_id : {$eq:1}}, {$pull: {editorial: "Atlantida"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Figura 84: Captura de Pantalla 52

Fuente .- Elaboración Propia

Luego de ejecutarse comando el arreglo 'editorial' tiene una nueva componente para el documento con _id=1.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.find({_id:1}).pretty()
{
    "_id" : 1,
    "titulo" : "El aleph",
    "autor" : "Borges",
    "editorial" : [
        "Siglo XXI",
        "Planeta"
    ],
    "precio" : 15,
    "cantidad" : 1
}
```

Figura 85: Captura de Pantalla 53

Fuente .- Elaboración Propia

- **updateMany ()**: permite actualizar un documento de manera masiva.

- Ejemplo por consola antes de actualizar un documento.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.find()
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1, "descripcion" : "" }
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 15, "cantidad" : 1 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
>
```

Figura 86: Captura de Pantalla 54

Fuente .- Elaboración Propia

- Ejemplo por consola de modificación masiva realiza la con todos los libros cuyo `_id` sea mayor a 2, fijando el campo `cantidad` con 0.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.updateMany( {_id : {$gt:2}}, {$set : {cantidad:0}})
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
>
```

Figura 87: Captura de Pantalla 55

Fuente .- Elaboración Propia

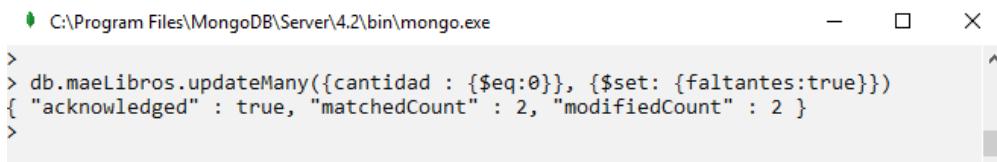
Luego de ejecutarse el método **UpdateMany()** se modificó la **Cantidad** con valor 0 cuyos y el campo `_id` mayor a 2

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.find({_id : {$gt:2}}).sort({_id:1}).pretty()
{
    "_id" : 3,
    "titulo" : "Aprenda PHP",
    "autor" : "Mario Molina",
    "editorial" : [
        "Siglo XXI",
        "Planeta"
    ],
    "precio" : 50,
    "cantidad" : 0
}
{
    "_id" : 4,
    "titulo" : "Java en 10 minutos",
    "editorial" : [
        "Siglo XXI"
    ],
    "precio" : 45,
    "cantidad" : 0,
    "descripcion" : ""
}
```

Figura 88: Captura de Pantalla 56

Fuente .- Elaboración Propia

La segunda modificación masiva se plantea a todos los libros que almacenan en el campo **Cantidad** el valor 0. Además, agregando el campo **Faltantes** a true.



```
> db.maeLibros.updateMany({cantidad : {$eq:0}}, {$set: {faltantes:true}})
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
>
```

Figura 89: Captura de Pantalla 57
Fuente .- Elaboración Propia

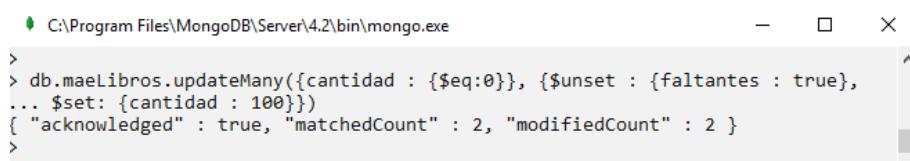
Luego de ejecutarse el método **updateMany()** se agregó el campo **Faltantes** con el valor true para todos los libros que almacenan en el campo **Cantidad** 0.



```
> db.maeLibros.find({cantidad : {$eq:0}}).sort({_id:1}).pretty()
{
  "_id" : 3,
  "titulo" : "Aprenda PHP",
  "autor" : "Mario Molina",
  "editorial" : [
    {
      "Siglo XXI",
      "Planeta"
    },
    {
      "precio" : 50,
      "cantidad" : 0,
      "faltantes" : true
    }
  ],
  "_id" : 4,
  "titulo" : "Java en 10 minutos",
  "editorial" : [
    {
      "Siglo XXI"
    },
    {
      "precio" : 45,
      "cantidad" : 0,
      "descripcion" : "",
      "faltantes" : true
    }
}
>
```

Figura 90: Captura de Pantalla 58
Fuente .- Elaboración Propia

La tercera modificación masiva se plantea a todos los libros que almacenan en el campo **Cantidad** el valor 0. Además, eliminamos el campo **Faltantes** y fijamos el campo **Cantidad** 100.



```
> db.maeLibros.updateMany({cantidad : {$eq:0}}, {$unset : {faltantes : true},
... $set: {cantidad : 100}})
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
>
```

Figura 91: Captura de Pantalla 59
Fuente .- Elaboración Propia

- Evidencia por consola del documento actualizado.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.find({_id : {$gt:2}}).sort({_id:1}).pretty()
{
    "_id" : 3,
    "titulo" : "Aprenda PHP",
    "autor" : "Mario Molina",
    "editorial" : [
        "Siglo XXI",
        "Planeta"
    ],
    "precio" : 50,
    "cantidad" : 100
}
{
    "_id" : 4,
    "titulo" : "Java en 10 minutos",
    "editorial" : [
        "Siglo XXI"
    ],
    "precio" : 45,
    "cantidad" : 100,
    "descripcion" : ""
}
>
```

Figura 92: Captura de Pantalla 60
Fuente .- Elaboración Propia

- **replaceOne ()**: permite actualizar el primer documento de acuerdo a un criterio.
- Ejemplo de reemplazo del primer libro con campo cantidad en 50. Además, otros documentos con los contenidos del **Título** “Java 10 Fundamentals”, **Autor** “Cay S. Horstmann”, **Editorial** “Planeta” y **Precio** 25.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
>
> db.maeLibros.replaceOne({cantidad : 50}, {"titulo: "Java 10 Fundamentals", autor: "Cay S. Horstmann", ... editorial : ["Planeta"], precio: 25})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> ■
```

Figura 93: Captura de Pantalla 61
Fuente .- Elaboración Propia

- Evidencia por consola del documento actualizado.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.maeLibros.find().sort({_id:1}).pretty()
{
    "_id" : 1,
    "titulo" : "Java 10 Fundamentals",
    "autor" : "Cay S. Horstmann",
    "editorial" : [
        "Planeta"
    ],
    "precio" : 25
}
{
    "_id" : 2,
    "titulo" : "Martin Fierro",
    "autor" : "Jose Hernandez",
    "editorial" : [
        "Siglo XXI"
    ],
    "precio" : 50,
    "cantidad" : 12
}
```

Figura 94: Captura de Pantalla 62
Fuente .- Elaboración Propia

3.1.4. DELETE

El último de las operaciones CRUD tiene métodos para eliminar uno o varios elementos.

Sintaxis:

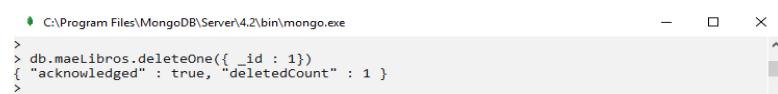
- db.<collection>.deleteOne(filter);
- db.<collection>.deleteMany(filter);

Donde:

- Filter: condición de eliminación.

A continuación, detallamos los comandos.

- **deleteOne ()**: permite actualizar el primer documento de acuerdo a un criterio.
- Ejemplo de eliminación de un documento **maeLibros** con `_id 1`.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.deleteOne({ '_id' : 1 })
{ "acknowledged" : true, "deletedCount" : 1 }
```

Figura 95: Captura de Pantalla 63

Fuente .- Elaboración Propia

- Ejemplo de eliminación de documento con `_id 1`.

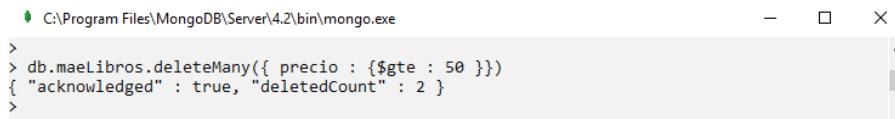


```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.deleteOne({ '_id' : {$eq : 1} })
```

Figura 96: Captura de Pantalla 64

Fuente .- Elaboración Propia

- **deleteMany ()**: elimina todos los registros que cumplan la condición.
- Ejemplo de eliminación de Libros que tiene un precio mayor o igual a 50.

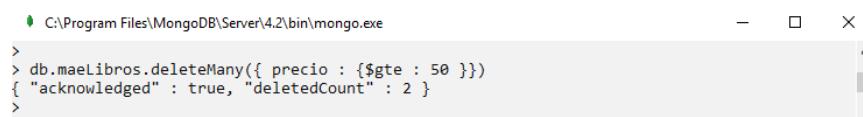


```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.deleteMany({ 'precio' : {$gte : 50} })
{ "acknowledged" : true, "deletedCount" : 2 }
```

Figura 97: Captura de Pantalla 65

Fuente .- Elaboración Propia

Nota: ambos métodos informan la cantidad de documentos afectados.



```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.maeLibros.deleteMany({ 'precio' : {$gte : 50} })
{ "acknowledged" : true, "deletedCount" : 2 }
```

Figura 98: Captura de Pantalla 66

Fuente .- Elaboración Propia

Resumen

1. El ingreso de documentos a una colección debemos realizarla con los métodos insertOne () y insertMany ().
2. La actualización de documentos de una colección debemos utilizar los métodos updateOne (), updateMany () y replaceOne ().
3. La eliminación de documentos de una colección debemos utilizar los métodos deleteOne () y deleteMany ().

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://www.youtube.com/watch?v=cW8racyXuvY>
- <https://www.youtube.com/watch?v=c8n6JsQuX2A>
- <https://www.youtube.com/watch?v=VMC6gSUo2IM>
- https://www.youtube.com/watch?v=m9e_aEORGbM



GESTIÓN DE DATOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno utiliza los métodos de gestión de datos con la base de datos MongoDB.

TEMARIO

4.1 Tema 8 : Consultas simples

- 4.1.1 : Find ()
- 4.1.2 : FindOne ()

4.2 Tema 9 : Consultas avanzadas

- 4.2.1 : Comparación
- 4.2.2 : Existencias
- 4.2.3 : Lógicas
- 4.2.4 : Arrrays
- 4.2.5 : Subdocumentos

4.3 Tema 10 : Curosres

- 4.3.1 : Count ()
- 4.3.2 : Sort ()
- 4.3.3 : Limit ()
- 4.3.4 : Skip ()
- 4.3.5 : toArray ()

ACTIVIDADES PROPUESTAS

- Los alumnos realizan las lecturas propuestas.
- Los alumnos realizan operaciones de consulta utilizando distintos operadores.

4.1. CONSULTAS SIMPLES

Una de las operaciones más comunes son las consultas de bases de datos, que a su vez constituirá la base de los informes. MongoDB tiene métodos de consulta:

Sintaxis:

- db.<collection>.find({filtro}, {proyección}).<agregación>()
- db.<collection>.findOne({filtro}, {proyección}).<agregación>()

Donde:

- Filtro: es un filtro por donde pasa los documentos de la colección.
- Proyección: muestra u ocultar la clave-valor del documento.
- Agregación: manipular el conjunto de resultados finales.

En caso no se especifique ningún argumento en el método **Find ()** se devuelve todos los documentos de la colección. Equivalente a **SELECT * FROM table** en una base de datos relacional.

```
> db.maeLibros.find()
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45,
"cantidad" : 100 }
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ],
"precio" : 15, "cantidad" : 1 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI",
"Planeta" ], "precio" : 50, "cantidad" : 100 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo
XXI" ], "precio" : 50, "cantidad" : 12 }
> -
```

Figura 99: Captura de Pantalla 67
Fuente ..- Elaboración Propia

El valor de retorno o salida del método **Find ()** no es el resultado en sí, sino un cursor o puntero hacia los recursos que se utiliza para extraer datos. El filtro es una expresión JSON, que define los criterios por los cuales los documentos se incluyen o se excluyen del conjunto de resultados finales. El filtro consta de dos elementos, un nombre de campo de documento y una expresión. La expresión puede ser un valor codificado, una expresión regular o una declaración que involucre uno o más operadores.

4.1.1. Find()

Devuelve los documentos de una colección o vista que coincidan con la consulta.

- Ejemplo de búsqueda por consola de personas con 34 años.

```
>
> db.people.find(
... {age : 34}
... ).pretty()
{
    "_id" : ObjectId("5da9ec705872d69dfd5f4d94"),
    "isActive" : false,
    "balance" : "$3,131.00",
    "picture" : "http://placehold.it/32x32",
    "age" : 34,
    "name" : "Morgan Cook",
    "company" : "Robocomm",
    "phone" : "816-455-37785",
    "email" : "morgan@robocomm.com",
    "about" : "Dolorum occaecat cupiditate cum ea"
}
```

Figura 100: Captura de Pantalla 68
Fuente .- Elaboración Propia

- Ejemplo de búsqueda por MongoDB Compass de personas con 34 años.



Figura 101: Captura de Pantalla 69
Fuente .- Elaboración Propia

- Ejemplo de búsqueda por consola de personas con 34 años y activo isActive.

```
> db.people.find(
... {age : 34, isActive : true}
... ).pretty()
{
    "_id" : ObjectId("5da9ec705872d69dfd5f4dac"),
    "isActive" : true,
    "balance" : "$1,066.00",
    "picture" : "http://placehold.it/32x32",
    "age" : 34,
    "name" : "Julia Young",
    "company" : "Genland",
    "phone" : "831-524-30334",
    "email" : "julia@google.com",
    "address" : "18282, ChulaVista, Bleeker Street",
    "about" : "Nisi excentetur. Lorem mollit amet amet aliqua non consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupiditate non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
}
```

Figura 102: Captura de Pantalla 70
Fuente .- Elaboración Propia

- Ejemplo de búsqueda por MongoDB Compass de personas con 34 años y activo isActive.

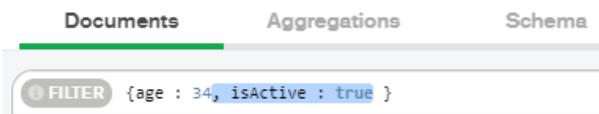


Figura 103: Captura de Pantalla 71
Fuente .- Elaboración Propia

Los resultados muestran todos los campos del documento, equivalente al SELECT * en una consulta de bases de datos relacionales. En caso se necesite seleccionar algunos campos, se utiliza el segundo parámetro de la consulta **find ()** para definir una proyección.

- Ejemplo de búsqueda por consola por name, age, isActive, _id:0.

```
> db.people.find(
... {age:34, isActive:true},
... {name:1, age:1, isActive:1, _id:0}
... ).pretty()
{
  "isActive" : true,
  "age" : 34,
  "name" : "Julia Young"
}
{
  "isActive" : true,
  "age" : 34,
  "name" : "Mackenzie Clapton"
}
{
  "isActive" : true,
  "age" : 34,
  "name" : "Destiny Calhoun"
}
{
  "isActive" : true,
  "age" : 34,
  "name" : "Amelia Carroll"
}
{
  "isActive" : true,
  "age" : 34,
  "name" : "Lauren Hailey"
}
{
  "isActive" : true,
  "age" : 34,
  "name" : "Molly Chapman"
}
{
  "isActive" : true,
  "age" : 34,
  "name" : "Leah Timmons"
}
```

Figura 104: Captura de Pantalla 72

Fuente .- Elaboración Propia

Nota: por defecto el `_id` siempre se muestra para ocultarlo hay que especificarlo en la proyección `_id:0`.

- Ejemplo de búsqueda por MongoDB Compass por name, age, isActive, `_id:0`.

The screenshot shows the MongoDB Compass interface with the 'Documents' tab selected. Below it, a series of operations are listed in a sequence:

- FILTER**: `{age : 34, isActive : true }`
- PROJECT**: `{name:1, age:1, isActive:1, _id:0}`
- SORT**
- COLLATION**

Figura 105: Captura de Pantalla 73

Fuente .- Elaboración Propia

- Ejemplo de búsqueda por consola quitando campos no deseados.

```
> db.people.find(
... {age:34, isActive:true},
... {name:0, age:0, isActive:0,_id:0}
... ).pretty()
{
  "balance" : "$1,066.00",
  "picture" : "http://placehold.it/32x32",
  "company" : "Genland",
  "phone" : "831-524-30334",
  "email" : "julia@genland.com",
  "address" : "18282, ChulaVista, Bleeker Street",
  "about" : "Nisi excepteur labore mollit amet sint al
```

Figura 106: Captura de Pantalla 74

Fuente .- Elaboración Propia

- Ejemplo de búsqueda por MongoDB Compass quitando campos no deseados.

The screenshot shows the MongoDB Compass interface with the 'Documents' tab selected. Below it, a series of operations are listed in a sequence:

- FILTER**: `{age : 34, isActive : true }`
- PROJECT**: `{name:0, age:0, isActive:0, _id:0}`

Figura 107: Captura de Pantalla 75

Fuente .- Elaboración Propia

4.1.2. FindOne()

Devuelve un documento que satisface los criterios de consulta especificados en la colección. Si varios documentos satisfacen la consulta, este método devuelve el primer documento según el orden natural de los documentos en disco. El método puede especificarse los mismos filtros que el método Find().

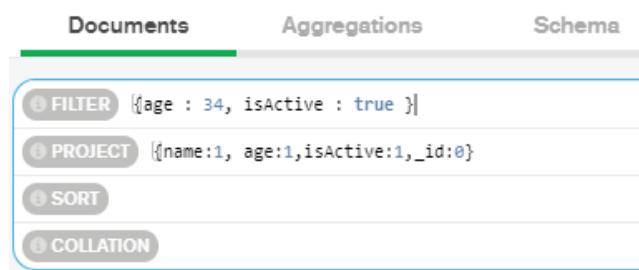


Figura 108: Captura de Pantalla 76
Fuente .- Elaboración Propia

4.2. CONSULTAS AVANZADAS

En la sección anterior se utilizó el **find ()** y **findOne ()** sin la inclusión de operadores para ejecutar consultas simples. A continuación, se explicará posibles operadores a incluirse en las consultas complejas.

Tabla 2
Operadores

Operador	Significado
\$gt	Coincide con valores mayores que un valor especificado.
\$gte	Coincide con valores mayores o iguales a un valor especificado.
\$lt	Coincide con valores inferiores a un valor especificado.
\$lte	Coincide con valores que son menores o iguales a un valor especificado.
\$in	Coincide con cualquiera de los valores especificados en una matriz.
\$eq	Coincide con valores que son iguales a un valor especificado.
\$ne	Coincide con todos los valores que no son iguales a un valor especificado.
\$nin	No coincide con ninguno de los valores especificados en una matriz.

Nota. Elaboración Propia

4.2.1. Comparación

\$gt (abreviatura de “greater than” en inglés)

- Ejemplo de búsqueda por consola de personas con más de 30 años.

```
db.people.find(
.. {age:{$gt:30}},
.. {name:1, age:1})
"_id" : ObjectId("5da9ec705872d69dfd5f4d87"), "age" : 36, "name" : "Julia Gate" }
"_id" : ObjectId("5da9ec705872d69dfd5f4d88"), "age" : 32, "name" : "Aubrey Brooks" }
"_id" : ObjectId("5da9ec705872d69dfd5f4d8d"), "age" : 33, "name" : "Riley Fulton" }
"_id" : ObjectId("5da9ec705872d69dfd5f4d90"), "age" : 36, "name" : "Isabella Carter" }
"_id" : ObjectId("5da9ec705872d69dfd5f4d91"), "age" : 36, "name" : "Audrey Campbell" }
"_id" : ObjectId("5da9ec705872d69dfd5f4d93"), "age" : 39, "name" : "Addison Sheldon" }
```

Figura 109: Captura de Pantalla 77
Fuente .- Elaboración Propia

- Ejemplo de búsqueda por MongoDB Compass de personas con más de 30 años.



Figura 110: Captura de Pantalla 78
Fuente .- Elaboración Propia

La consulta tiene el primer parámetro al método Find () y una proyección como segundo parámetro (name y age).

\$gte (abreviatura de “greater than equals” en inglés)

- Ejemplo de búsqueda por consola de personas con 30 a más años.

```
> db.people.find(
... {age:{$gte:30}},
... {name:1, age:1})
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d87"), "age" : 36, "name" : "Julia Gate" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d88"), "age" : 32, "name" : "Aubrey Brooks" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d89"), "age" : 33, "name" : "Riley Fulton" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d8e"), "age" : 30, "name" : "Ariana Carlson" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d90"), "age" : 36, "name" : "Isabella Carter" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d91"), "age" : 36, "name" : "Audrey Campbell" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d93"), "age" : 39, "name" : "Addison Sheldon" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d94"), "age" : 34, "name" : "Morgan Cook" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d95"), "age" : 37, "name" : "Sarah Walkman" }
```

Figura 111: Captura de Pantalla 79

Fuente .- Elaboración Propia

- Ejemplo de búsqueda por MongoDB Compass de personas con 30 a más años.

The screenshot shows the MongoDB Compass interface with the 'Documents' tab selected. Below the tabs, there are two stages in a pipeline: 'FILTER' and 'PROJECT'. The 'FILTER' stage contains the query '{age:{\$gte:30}}'. The 'PROJECT' stage contains the projection '{name:1, age:1}'.

Figura 112: Captura de Pantalla 80

Fuente .- Elaboración Propia

La consulta tiene el primer parámetro al método **Find ()** y una proyección como segundo parámetro (name y age).

\$lt (abreviatura de “lower than” en inglés)

- Ejemplo de búsqueda por consola de personas de menos 30 de años.

```
> db.people.find(
... {age:{$lt:30}},
... {name:1, age:1})
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d82"), "age" : 23, "name" : "Hailey Timmons" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d83"), "age" : 25, "name" : "Sophia Oliver" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d84"), "age" : 23, "name" : "Melanie Bush" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d85"), "age" : 29, "name" : "Brianna Osborne" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d86"), "age" : 26, "name" : "Mariah Gilmore" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d89"), "age" : 20, "name" : "Isabella Thornton" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d8a"), "age" : 20, "name" : "Lillian Campbell" }
```

Figura 113: Captura de Pantalla 81

Fuente .- Elaboración Propia

- Ejemplo de búsqueda por MongoDB Compass de personas de menos 30 años.

The screenshot shows the MongoDB Compass interface with the 'Documents' tab selected. Below the tabs, there are two stages in a pipeline: 'FILTER' and 'PROJECT'. The 'FILTER' stage contains the query '{age:{\$lt:30}}'. The 'PROJECT' stage contains the projection '{name:1, age:1}'.

Figura 114: Captura de Pantalla 82

Fuente .- Elaboración Propia

\$lte (abreviatura de “lower than equals” en inglés)

Se aplica la misma estructura del comando anterior, cambiándolo a **\$lte**.

\$in (include in array)

- Ejemplo de búsqueda por consola de edad 25, 30 y 35 años.

```
> db.people.find(
... {age : {$in : [20, 30, 35]}},
... {name:1, age:1})
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d89"), "age" : 20, "name" : "Isabella Thornton" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d8a"), "age" : 20, "name" : "Lillian Campbell" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d8e"), "age" : 30, "name" : "Ariana Charlson" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d9f"), "age" : 35, "name" : "Brianna Chesterton" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4da4"), "age" : 35, "name" : "Destiny WifKinson" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4da6"), "age" : 35, "name" : "Amelia Vaughan" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4db0"), "age" : 30, "name" : "Makayla Fisher" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4dd0"), "age" : 35, "name" : "Sophie Freeman" }
```

Figura 115: Captura de Pantalla 83

Fuente .- Elaboración Propia

- Ejemplo de búsqueda por MongoDB Compass de edad 25, 30 y 35 años.

Figura 116: Captura de Pantalla 84

Fuente .- Elaboración Propia

4.2.2. Existencias

MongoDB es una base de datos sin esquema, es decir, los documentos siendo la misma colección pueden tener distintos campos. Incluso distintos tipos en cada documento.

Tabla 3
Operadores

Operador	Significado
\$exists	Coincide con documentos que tienen el campo especificado.
\$type	Selecciona documentos si un campo es del tipo especificado.

Nota. Elaboración Propia

Dicho esto, en ocasiones es útil realizar consultas que devuelva los documentos en los que exista realmente un determinado campo.

\$exists

- Ejemplo de consola existencia del campo **Company** en documento.

```
> db.people.find(
... {company:{$exists:true}},
... {name:1, age:1, company:1})
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d82"), "age" : 23, "name" : "Hailey Timmons",
"company" : "US Omniphotik" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d83"), "age" : 25, "name" : "Sophia Oliver", "company" : "Technogra" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d84"), "age" : 23, "name" : "Melanie Bush", "company" : "Steganographic" }
```

Figura 117: Captura de Pantalla 85

Fuente .- Elaboración Propia

- Ejemplo de MongoDB Compass existencia del campo **Company** en documento.

Figura 118: Captura de Pantalla 86

Fuente .- Elaboración Propia

Si necesitas buscar los documentos que NO tienen el campo **Company**, bastará con cambiar el **true** por un **false** en el **\$exists**.

\$type

MongoDB puede almacenar documentos con distintos tipos del mismo campo.

- Ejemplo: aunque **age** es un número para los documentos. Se puede insertar un documento nuevo con un string en campo. Por tanto, puede necesitarse filtrar por un campo determinado por el tipo. Esto se hace con el operador **\$type**.

Desde consola:

```
> db.people.find(
... {company : {$type: 2}},
... {name:1, age:1, company:1})
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d82"), "age" : 23, "name" : "Hailey Timmons",
"company" : "US Omniphotik" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d83"), "age" : 25, "name" : "Sophia Oliver", "company" : "Technogra" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d84"), "age" : 23, "name" : "Melanie Bush", "company" : "Steganographic" }
{ "_id" : ObjectId("5da9ec705872d69dfd5f4d85"), "age" : 29, "name" : "Brianna Osborne",
"company" : "Titanograf" }
```

Figura 119: Captura de Pantalla 87

Fuente .- Elaboración Propia

Desde MongoDB Compass:

Figura 120: Captura de Pantalla 88

Fuente .- Elaboración Propia

En este caso, busca los documentos cuyo campo company sea tipo 2=string.

A partir de MongoDB 3.6, el operador **\$type** acepta alias para los tipos BSON, además de los números correspondientes a los tipos BSON.

Tabla 4
Operador \$type

Type	Number	Alias	Notes
Double	1	“double”	
String	2	“string”	
Object	3	“object”	
Array	4	“array”	
Binary data	5	“binData”	
Undefined	6	“undefined”	Obsoleto
ObjectId	7	“objectId”	
Boolean	8	“bool”	
Date	9	“date”	
Null	10	“null”	
Regular Expression	11	“regex”	
DBPointer	12	“dbPointer”	Obsoleto
JavaScript	13	“javascript”	
Symbol	14	“symbol”	Obsoleto
JavaScript (with scope)	15	“javascriptWithScope”	
32-bit integer	16	“int”	
Timestamp	17	“timestamp”	
64-bit integer	18	“long”	
Decimal 128	19	“decimal”	New versión 3.4
Min key	-1	“minKey”	
Max key	127	“maxKey”	

Nota. Elaboración Propia

4.2.3. Lógicas

En bases de datos relacionales es muy típico añadir el operador OR al WHERE. Por ejemplo, WHERE gender = “female” OR age > 20. Hasta ahora las consultas que hemos revisado buscaban por uno o más campos, pero lo hacían con la lógica AND.

\$or

Desde consola:

```
>
> db.people.find(
... { $or: [{gender:"female"}, {age:{$gt:20}}]},
... {name:1, gender:1, age:1})
{ "_id" : ObjectId("5da9ec705872d69df5f4d82"), "age" : 23, "name" : "Hailey Timmons",
"gender" : "female" }
{ "_id" : ObjectId("5da9ec705872d69df5f4d83"), "age" : 25, "name" : "Sophia Oliver", "gender" : "female" }
{ "_id" : ObjectId("5da9ec705872d69df5f4d84"), "age" : 23, "name" : "Melanie Bush", "gender" : "female" }
{ "_id" : ObjectId("5da9ec705872d69df5f4d85"), "age" : 29, "name" : "Brianna Osborne", "gender" : "male" }
```

Figura 121: Captura de Pantalla 89

Fuente .- Elaboración Propia

Desde MongoDB Compass:

```
Documents Aggregations Schema
FILTER { $or: [{gender:"female"}, {age:{$gt:20}}] }
PROJECT { name:1, gender:1, age:1 }
```

Figura 122: Captura de Pantalla 90

Fuente .- Elaboración Propia

\$and

Se aplica la misma estructura del comando anterior, cambiándolo a **\$and**.

\$or y \$and

También pueden utilizarse juntos para filtrar datos. Por ejemplo: buscar en la colección “people” las personas cuya edad es mayor que 30 o cuyo género es “female” y su edad mayor que 50.

Desde consola:

```
>
> db.people.find(
... { $or : [ {age:{$gt:30}}, {$and : [{age:{$gt:50}}, {gender:"female"}]} ] })
```

Figura 123: Captura de Pantalla 91

Fuente .- Elaboración Propia

Desde MongoDB Compass:

```
Documents Aggregations Schema Explain Plan
FILTER { $or: [{age:{$gt:30}}, { $and: [{age:{$gt:50}}, {gender:"female"}]}] }
PROJECT { name:1, gender:1, age:1 }
```

Figura 124: Captura de Pantalla 92

Fuente .- Elaboración Propia

\$not

Busca los documentos que no cumplan una determinada condición.

Desde consola:

```
>
> db.people.find( {age: {$not: {$gt:30}}} )-
```

Figura 125: Captura de Pantalla 93
Fuente .- Elaboración Propia

Desde MongoDB Compass:

Figura 126: Captura de Pantalla 94
Fuente .- Elaboración Propia

4.2.4. Arrays

Se ha explicado en anteriores capítulos, MongoDB puede guardar array de elementos y pueden ser de cualquier tipo, es decir, string, números, arrays entre otros. En los datos de la base de datos **DBTest**, cada persona de la colección “people” tiene asociado un campo tags, que es un array de strings. Buscar un solo elemento dentro de ese array se necesita ejecutar el siguiente comando.

\$tag

Desde consola:

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

Figura 127: Captura de Pantalla 95
Fuente .- Elaboración Propia

Desde MongoDB Compass:

Figura 128: Captura de Pantalla 96
Fuente .- Elaboración Propia

\$all

Busca varios elementos dentro de un array

Por ejemplo: buscar todas las personas que contengan “laborum” y “sunt” en el campo tags. Devolver los documentos que contengan ambos valores. De esta manera, se ha especificado dos valores, pero se puede añadir todos los que se necesiten y solo se devolverán los documentos que los incluyan.

Desde consola:

```
> db.people.find({tags:{$in:["laborum", "sunt", "nisi"]}},  
... {name:1, tags:1})■
```

Figura 129: Captura de Pantalla 97

Fuente .- Elaboración Propia

Desde MongoDB Compass:

Figura 130: Captura de Pantalla 98

Fuente .- Elaboración Propia

\$in

De forma similar a la búsqueda anterior se puede devolver los documentos que contengan al menos uno de los elementos con el comando \$in.

Desde consola:

```
>  
> db.people.find({tags:{$in:["laborum", "sunt", "nisi"]}},  
... {name:1, tags:1})■
```

Figura 131: Captura de Pantalla 99

Fuente .- Elaboración Propia

Desde MongoDB Compass:

Figura 132: Captura de Pantalla 100

Fuente .- Elaboración Propia

\$nin

Se aplica la misma estructura del comando anterior, cambiándolo a **\$nin** y devolverá los no contenidos en el array.

\$size

Se aplica la misma estructura del comando anterior, **\$size** devuelve todos los documentos que tengan la cantidad de arreglos especificados.

Desde consola:

```
>
> db.people.find(tags:{$size:3})
```

Figura 133: Captura de Pantalla 101
Fuente .- Elaboración Propia

Desde MongoDB Compass:



Figura 134: Captura de Pantalla 102
Fuente .- Elaboración Propia

\$slice

Se aplica la misma estructura del comando anterior, **\$slice** devuelve todos los documentos que tengan un número determinado de arreglos.

Desde consola:

```
>
> db.people.find({tags:{$in:["laborum", "sunt", "nisi"]}}, ...
... {tags:{$slice:3}, name:1})
```

Figura 135: Captura de Pantalla 103
Fuente .- Elaboración Propia

Desde MongoDB Compass:

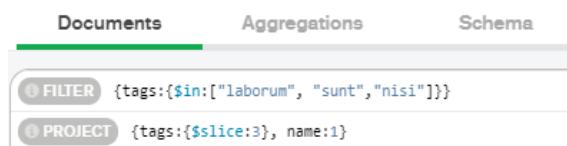


Figura 136: Captura de Pantalla 104
Fuente .- Elaboración Propia

Con la consulta anterior se devuelven los documentos filtrados, pero solo dos campos: los tres primeros elementos del array tags y el nombre de la persona. Para devolver los tres últimos, bastaría con usar un número negativo.

4.2.5. Subdocumentos

En los datos de ejemplo existe un campo llamado Friends, contiene un array de subdocumentos. Si necesitamos buscar los elementos del subdocumento por el **id 1** y el **nombre “Trinity Ford”** la consulta sería:

Desde consola:

```
> db.people.find({friends: {id:1, name: "Trinity Ford"}})_
```

Figura 137: Captura de Pantalla 105
Fuente .- Elaboración Propia

Desde MongoDB Compass:



Figura 138: Captura de Pantalla 106
Fuente .- Elaboración Propia

Además, podemos incluir comillas para buscar en un campo específico. De esta forma, MongoDB devolverá los subdocumentos que cumplen la condición.

En consola MongoDB:

```
> db.people.find({"friends.name" : "Trinity Ford"})_
```

Figura 139: Captura de Pantalla 107
Fuente .- Elaboración Propia

En MongoDB Compass:



Figura 140: Captura de Pantalla 108
Fuente .- Elaboración Propia

4.3. CURSORES

Cuando se realiza una consulta en MongoDB, el servidor devuelve un objeto cursor. Un objeto cursor es un iterador de resultados, es decir, la información reside en la memoria del servidor y el cliente accede al segmento de memoria mediante un puntero.

4.3.1. Count ()

Retorna el número de documentos de una colección o vista.

Desde consola:

```
> db.people.find({"friends.2.name":{$gte:"T"}}).count()
34
>
```

Figura 141: Captura de Pantalla 109
Fuente .- Elaboración Propia

Desde MongoDB Compass:

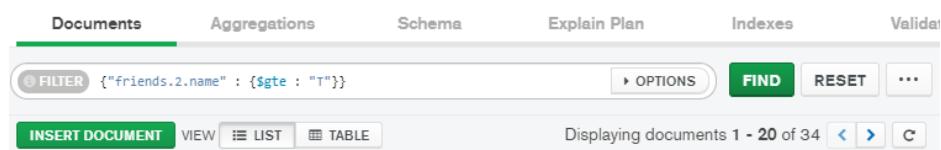


Figura 142: Captura de Pantalla 110
Fuente .- Elaboración Propia

4.3.2. Sort ()

Especifica el orden en el que la consulta devuelve documentos.

Desde consola:

```
>
> db.people.find({"friends.2.name" : {$gte:"T"}}, 
... {_id:0, name:1}).sort({name:1})
{ "name" : "Alexa Smith" }
{ "name" : "Alexis Gibbs" }
{ "name" : "Andrea Gerald" }
{ "name" : "Angelina Freeman" }
{ "name" : "Ariana Cramer" }
{ "name" : "Ava Wainwright" }
{ "name" : "Brooke Stanley" }
```

Figura 143: Captura de Pantalla 111
Fuente .- Elaboración Propia

Desde MongoDB Compass:

The screenshot shows the MongoDB Compass interface with the 'Aggregations' tab selected. The pipeline consists of the following stages:

- FILTER**: `{"friends.2.name" : {$gte : "T"}}`
- PROJECT**: `{_id:0, name:1}`
- SORT**: `{name : 1}`
- MAXTIMEMS**: 5000
- SKIP**: 0
- LIMIT**: 0

Figura 144: Captura de Pantalla 112
Fuente .- Elaboración Propia

Puede especificarse más de un campo separándolo por comas.

4.3.3. Limit ()

Especifica el número de resultados devueltos. Devolver los primero 5 registros.

Desde consola:

```
>
> db.people.find({"friends.2.name" : {$gte:"T"}},
... {_id:0, name:1}).sort({name:1})
{ "name" : "Alexa Smith" }
{ "name" : "Alexis Gibbs" }
{ "name" : "Andrea Gerald" }
{ "name" : "Angelina Freeman" }
{ "name" : "Ariana Cramer" }
{ "name" : "Ava Wainwright" }
{ "name" : "Brooke Stanley" }
```

Figura 145: Captura de Pantalla 113
Fuente .- Elaboración Propia

Desde MongoDB Compass:

The screenshot shows the MongoDB Compass interface with the 'Aggregations' tab selected. The pipeline consists of the following stages:

- FILTER**: `{"friends.2.name" : {$gte : "T"}}`
- PROJECT**: `{_id:0, name:1}`
- SORT**: `{name : 1}`
- MAXTIMEMS**: 5000
- SKIP**: 0
- LIMIT**: 5

Figura 146: Captura de Pantalla 114
Fuente .- Elaboración Propia

4.3.4. Skip ()

Ignora los “N” registros especificados. No devolver los primero 5 registros.

Desde consola:

```
>
> db.people.find({"friends.2.name" : {$gte:"T"}},
... {_id:0, name:1}).sort({name:1})
{ "name" : "Alexa Smith" }
{ "name" : "Alexis Gibbs" }
{ "name" : "Andrea Gerald" }
{ "name" : "Angelina Freeman" }
{ "name" : "Ariana Cramer" }
{ "name" : "Ava Wainwright" }
{ "name" : "Brooke Stanley" }
```

Figura 147: Captura de Pantalla 115

Fuente .- Elaboración Propia

Desde MongoDB Compass:

Figura 148: Captura de Pantalla 116

Fuente .- Elaboración Propia

4.3.5. toArray()

Almacena los resultados en un array que puede asignar a una variable.

Desde consola:

```
>
> db.people.find({"friends.2.name" : {$gte:"T"}},
... {_id:0, name:1}).sort({name:1})
{ "name" : "Alexa Smith" }
{ "name" : "Alexis Gibbs" }
{ "name" : "Andrea Gerald" }
{ "name" : "Angelina Freeman" }
{ "name" : "Ariana Cramer" }
{ "name" : "Ava Wainwright" }
{ "name" : "Brooke Stanley" }
```

Figura 149: Captura de Pantalla 117

Fuente .- Elaboración Propia

```
>
> db.people.find({"friends.2.name" : {$gte:"T"}},
... {_id:0, name:1}).sort({name:1})
{ "name" : "Alexa Smith" }
{ "name" : "Alexis Gibbs" }
{ "name" : "Andrea Gerald" }
{ "name" : "Angelina Freeman" }
{ "name" : "Ariana Cramer" }
{ "name" : "Ava Wainwright" }
{ "name" : "Brooke Stanley" }
```

Figura 150: Captura de Pantalla 118

Fuente .- Elaboración Propia

Desde MongoDB Compass:

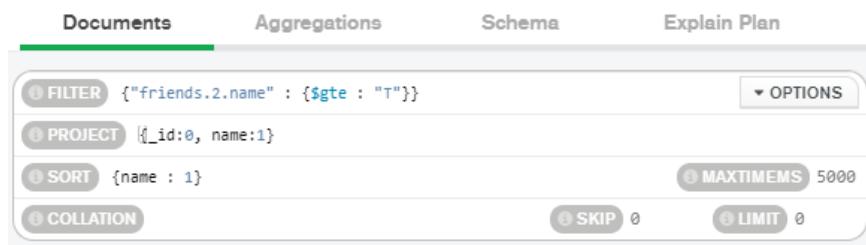


Figura 151: Captura de Pantalla 119
Fuente .- Elaboración Propia

Resumen

1. Un SGBD es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información de una base de datos.
2. Las bases de datos NoSQL no cumplen con el esquema entidad-relación.
3. El teorema CAP dice que en sistemas distribuidos (NoSQL) es imposible garantizar a la vez:
 - Consistencia
 - Disponibilidad
 - Tolerancia a particiones

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://www.youtube.com/watch?v=QMrTgeSICe4>
- <https://www.youtube.com/watch?v=8gbDA7kH5zg>
- <https://www.youtube.com/watch?v=gwhmz6f49nc&t=3s>
- <https://www.youtube.com/watch?v=L9yl9wYt2IY>



AGGREGATION FRAMEWORK

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno realizar cálculos y agrupación de resultados a los documentos almacenados.

TEMARIO

5.1 Tema 11 : Operadores simples

- 5.1.1 : \$project
- 5.1.2 : \$match
- 5.2.3 : \$group
- 5.2.4 : \$sort
- 5.2.5 : \$limit
- 5.2.6 : \$skip

5.2 Tema 12 : Operadores de expresión

- 5.2.1 : Agrupación
- 5.2.2 : Aritméticos
- 5.2.3 : Comparación
- 5.2.4 : Booleanos
- 5.2.5 : Condicionales
- 5.2.6 : Cadenas
- 5.2.7 : Fechas

ACTIVIDADES PROPUESTAS

- Los alumnos realizan las lecturas propuestas.
- Los alumnos realizan operaciones de consulta a MongoDB.

5.1. OPERADORES SIMPLES

5.1.1. \$project

Este **pipeline** tiene una funcionalidad equivalente al **SELECT** en una consulta SQL en una base de datos relacional. Igual que en SQL se puede modificar los campos del documento añadiendo otros, eliminándolos, renombrarlo o adicionando campos calculados.

Desde consola:

```
> db.people.aggregate({
...   $project: {
...     isActive: 1,
...     company: 1,
...     name: 1,
...     age: 1,
...     addedAge: {
...       $add: [
...         "$age",
...         10
...       ]
...     },
...     upperName: {
...       $toUpperCase: "$name"
...     }
...   }
...});
```

Figura 152: Captura de Pantalla 120
Fuente .- Elaboración Propia

Desde MongoDB Compass:



Figura 153: Captura de Pantalla 121
Fuente .- Elaboración Propia

En este caso, la colección people se crea dos campos: **addedAge** igual al valor del campo **age** pero incrementado en 10, utilizando el operador **\$add**. Por otro lado, **upperName** igual al valor del campo **name**, pero con el texto en mayúscula, utilizando el operador **\$toUpperCase**.

5.1.2. \$match

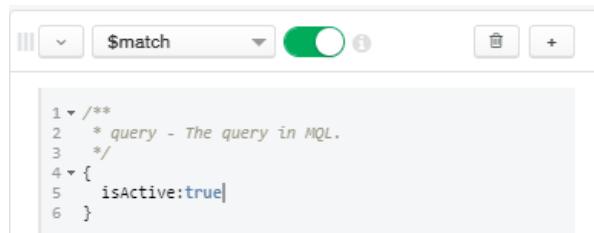
El pipeline **\$match** se utiliza para filtrar los documentos que se pasarán al siguiente pipeline de la consulta con Aggregation Framework.

Desde consola:

```
db.people.aggregate({ $match: { isActive:true}})
```

Figura 154: Captura de Pantalla 122
Fuente .- Elaboración Propia

Desde MongoDB Compass:



```
1 /**
2  * query - The query in MQL.
3 */
4 {
5   isActive:true
6 }
```

Figura 155: Captura de Pantalla 123
Fuente .- Elaboración Propia

De esta manera, se está filtrando personas cuyo campo **isActive** es true. En caso se combine **\$match** y **\$project**, la sentencia será más eficiente al colocar **\$match** primero, ya que **\$project** tendrá que procesar muchos menos documentos.

Desde consola:

```
> db.people.aggregate(
... {$match: {isActive:true}},
... {$project: {isActive:1, name:1, mainAddress:"address.primary"}}
... )
```

Figura 156: Captura de Pantalla 124
Fuente .- Elaboración Propia

Desde MongoDB Compass:



```
1 /**
2  * specifications - The fields to
3  * include or exclude.
4 */
5 {
6   isActive:1, name:1, mainAddress:"$address.primary"
7 }
```

Figura 157: Captura de Pantalla 125
Fuente .- Elaboración Propia

5.1.3. \$group

Hasta el momento hemos revisado funciones para filtrar documentos y mostrar solo campos deseados. Ahora con el pipeline **\$group** podemos agruparlos.

Desde consola:

```
db.people.aggregate({ $group: { _id: "$age"}})
```

Figura 158: Captura de Pantalla 126
Fuente .- Elaboración Propia

Nota: `$group` siempre debe tener un campo “`_id`”. Es el campo por el que vas a agrupar los resultados, así que, lógicamente, es obligatorio usarlo. En ejercicio anterior se agrupó por el campo `age`. Si quisieramos agrupar por varios campos, debemos utilizar una consulta similar.

Desde MongoDB Compass:

```
1 /**
2  * _id - The id of the group.
3  * field1 - The first field name.
4 */
5 {
6   _id: { age: "$age" }
7 }
```

Figura 159: Captura de Pantalla 127

Fuente .- Elaboración Propia

Adicionalmente, podemos adicionarle al `$group` los operadores: `$sum`, `$avg`, `$max`

Desde consola:

```
db.people.aggregate({
.. $group: {
..   _id: { gender: "$gender" },
..   average: { $avg: "$age" },
..   count: { $sum: 1 }
.. }})
```

Figura 160: Captura de Pantalla 128

Fuente .- Elaboración Propia

Desde MongoDB Compass:

```
1 /**
2  * _id - The id of the group.
3  * field1 - The first field name.
4 */
5 {
6   _id: { gender: "$gender" },
7   average: { $avg: "$age" },
8   count: { $sum: 1 }
9 }
```

Figura 161: Captura de Pantalla 129

Fuente .- Elaboración Propia

En este ejemplo, se agrupó por el campo `gender`, y una vez agrupados los datos, se muestra la edad media de cada género y la suma separada de hombres y mujeres.

```
> db.people.aggregate({
... $group: {
...   _id: { gender: "$gender" },
...   average: { $avg: "$age" },
...   count: { $sum: 1 }
... }})
{ "_id" : { "gender" : "male" }, "average" : 31.043478260869566, "count" : 46 }
{ "_id" : { "gender" : "female" }, "average" : 29.074074074074073, "count" : 54 }
```

Figura 162: Captura de Pantalla 130

Fuente .- Elaboración Propia

Podemos combinar las 3 funciones **\$project**, **\$match** y **\$group**.

Desde Consola:

```
db.people.aggregate(
.. { $match: { isActive: true } },
.. { $group: { _id: { gender: "$gender", age: "$age" },
..             count: { $sum: 1 }
..           },
.. { $project: { _id: 0, type: "$_id", total: "$count" } }
.. )
```

Figura 163: Captura de Pantalla 131
Fuente .- Elaboración Propia

Desde MongoDB Compass:

The screenshot shows the MongoDB Compass interface with three stacked aggregation pipeline stages:

- \$match:** A stage where documents are filtered based on the condition `isActive: true`.
- \$group:** A stage where documents are grouped by `_id: { gender: "$gender", age: "$age" }` and the count of documents in each group is calculated using the `$sum` operator.
- \$project:** A stage where the resulting fields are projected as `_id: 0, type: "$_id", total: "$count"`.

Figura 164: Captura de Pantalla 132
Fuente .- Elaboración Propia

Podemos observar el filtro de documentos mostrando solo aquellos cuyo campo **isActive** es true. Luego, se está agrupando por **gender** y **age**, y calculando el total de cada rango, es decir, cuántas personas del género femenino tienen 32 años, cuántas del género masculino tienen 18 y así sucesivamente. Para finalizar, se muestra el conjunto de resultados deseados, que es el campo **_id** renombrado a **type** y el campo **count** renombrado a **total**.

```
{
  "type" : { "gender" : "female", "age" : 30 }, "total" : 2
  "type" : { "gender" : "male", "age" : 35 }, "total" : 1
  "type" : { "gender" : "female", "age" : 27 }, "total" : 2
  "type" : { "gender" : "male", "age" : 38 }, "total" : 1
  "type" : { "gender" : "female", "age" : 24 }, "total" : 2
  "type" : { "gender" : "female", "age" : 20 }, "total" : 1
  "type" : { "gender" : "male", "age" : 24 }, "total" : 1
  "type" : { "gender" : "female", "age" : 34 }, "total" : 3
  "type" : { "gender" : "male", "age" : 25 }, "total" : 1
  "type" : { "gender" : "female", "age" : 26 }, "total" : 3
  "type" : { "gender" : "female", "age" : 32 }, "total" : 1
  "type" : { "gender" : "female", "age" : 35 }, "total" : 2
  "type" : { "gender" : "male", "age" : 21 }, "total" : 1
}
```

Figura 165: Captura de Pantalla 133

Fuente .- Elaboración Propia

5.1.4. \$sort

Permite realizar una agregación de los resultados.

Desde Consola:

```
db.people.aggregate(
  . { $sort: { age: 1 } }
  . )
```

Figura 166: Captura de Pantalla 134

Fuente .- Elaboración Propia

Desde MongoDB Compass:



Figura 167: Captura de Pantalla 135

Fuente .- Elaboración Propia

Con esta consulta se devuelven los datos ordenados por edad, en orden ascendente. Si quisieras el orden inverso bastaría con escribir un -1 en lugar de un 1 en la consulta.

¿Qué pasa cuando los tipos de campos son diferentes?

Como bien lo hemos indicado, MongoDB no tiene un esquema definido, lo que en este caso quiere decir que el campo **age** puede ser un número, una cadena de caracteres o incluso una fecha. Es más, mientras que en un documento el valor de **age** puede ser un número, en otro puede ser un **String**. En este contexto, MongoDB seguirá funcionando de igual manera. Cuando se trata con casos así, debe considerarse los tipos de datos tienen prioridad sobre otros según la lista siguiente:

- Null
- Valores numéricos (int, long, double)
- Cadena de caracteres
- Objetos
- Arrays

- Datos binarios
- ObjectId (como el campo _id que MongoDB genera para cada documento)
- Boolean
- Fechas
- Expresiones regulares

5.1.5. \$limit

Especifica el número de documentos devueltos de la consulta.

Desde Consola:

```
db.people.aggregate( { $limit: 5 } )
```

Figura 168: Captura de Pantalla 136
Fuente .- Elaboración Propia

Desde MongoDB Compass:

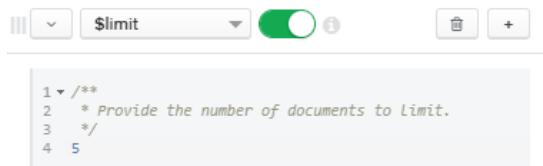


Figura 169: Captura de Pantalla 137
Fuente .- Elaboración Propia

5.1.6. \$skip

Especifica el número de documentos que descarta en una consulta.

Desde Consola:

```
db.people.aggregate( { $skip : 2 } )
```

Figura 170: Captura de Pantalla 138
Fuente .- Elaboración Propia

Desde MongoDB Compass:

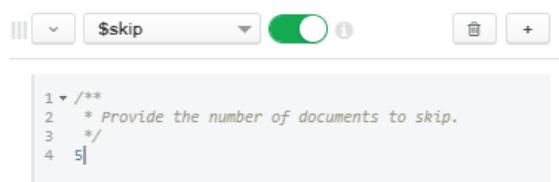


Figura 171: Captura de Pantalla 139
Fuente .- Elaboración Propia

5.2. OPERADORES DE EXPRESIÓN

5.2.1. Agrupación

Los operadores de agrupación son aquellos que se utilizan al momento de agrupar información con el pipeline \$group.

Tabla 5
Operadores de Agrupación

Operador	Descripción
\$first	devuelve el primer valor de un campo en un grupo. Si el grupo no está ordenado, el valor mostrado será impredecible.
\$last	devuelve el último valor de un campo en un grupo. Si el grupo no está ordenado, el valor mostrado será impredecible.
\$max	devuelve el valor más alto de un determinado campo dentro un grupo.
\$min	devuelve el valor más pequeño de un determinado campo dentro de un grupo.
\$avg	calcula la media aritmética de los valores dentro del campo especificado y los devuelve.
\$sum	suma todos los valores de un campo y los devuelve.

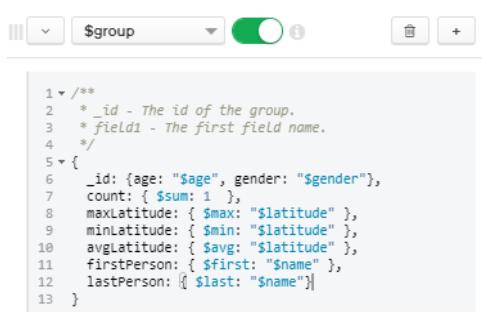
Nota. Elaboración Propia

Desde consola:

```
db.people.aggregate(
.. { $sort: { name: 1 } },
.. { $group: { _id: { age: "$age", gender: "$gender" },
..   count: { $sum: 1 },
..   maxLatitude: { $max: "$latitude" },
..   minLatitude: { $min: "$latitude" },
..   avgLatitude: { $avg: "$latitude" },
..   firstPerson: { $first: "$name" },
..   lastPerson: { $last: "$name" }
..   }
.. })
```

Figura 172: Captura de Pantalla 140
Fuente .- Elaboración Propia

Desde MongoDB Compass.



```
1 /**
2  * _id - The id of the group.
3  * fieldId - The first field name.
4 */
5 {
6   _id: { age: "$age", gender: "$gender" },
7   count: { $sum: 1 },
8   maxLatitude: { $max: "$latitude" },
9   minLatitude: { $min: "$latitude" },
10  avgLatitude: { $avg: "$latitude" },
11  firstPerson: { $first: "$name" },
12  lastPerson: { $last: "$name" }
13 }
```

Figura 173: Captura de Pantalla 141
Fuente .- Elaboración Propia

5.2.2. Aritméticos

Los operadores de agrupación tienen la limitación de que solo pueden usarse con el pipeline **\$group**. En cambio, los operadores aritméticos pueden utilizarse con **\$project**.

Tabla 6
Operadores Aritméticos

Operador	Descripción
\$add	realiza la suma de un array de números.
\$divide	divide dos números.
\$mod	a partir de 2 números se calcula el resto producido al dividir el primero entre el segundo.
\$multiply	multiplica dos números.
\$subtract	a partir de dos números realiza la resta.

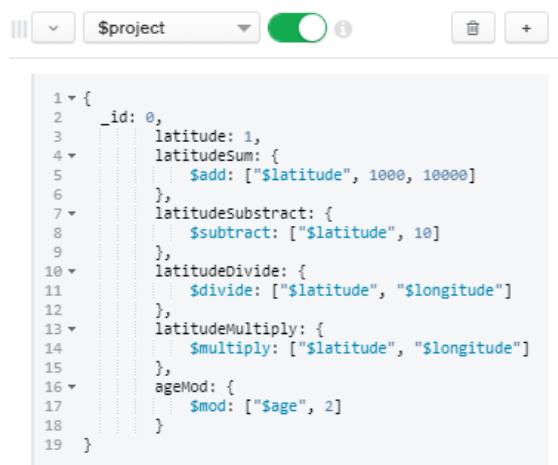
Nota. Elaboración Propia

Desde consola:

```
db.people.aggregate({
  ... $project: { _id: 0, latitude: 1,
    ...           latitudeSum: { $add: [ "$latitude", 1000, 10000 ] },
    ...           latitudeSubtract: { $subtract: [ "$latitude", 10 ] },
    ...           latitudeDivide: { $divide: [ "$latitude", "$longitude" ] },
    ...           latitudeMultiply: { $multiply: [ "$latitude", "$longitude" ] },
    ...           ageMod: { $mod: [ "$age", 2 ] }
  ... })
... })
```

Figura 174: Captura de Pantalla 142
Fuente .- Elaboración Propia

Desde MongoDB Compass.



```
1 ▾ {
2   _id: 0,
3   latitude: 1,
4   latitudeSum: {
5     $add: ["$latitude", 1000, 10000]
6   },
7   latitudeSubtract: {
8     $subtract: ["$latitude", 10]
9   },
10  latitudeDivide: {
11    $divide: ["$latitude", "$longitude"]
12  },
13  latitudeMultiply: {
14    $multiply: ["$latitude", "$longitude"]
15  },
16  ageMod: {
17    $mod: ["$age", 2]
18  }
19 }
```

Figura 175: Captura de Pantalla 143
Fuente .- Elaboración Propia

5.2.3. Comparación

Este tipo de operadores se utilizan para comparar valores y devolver un resultado. (Faci, 2019).

Tabla 7
Operadores de Comparación

Operador	Descripción
\$cmp	compara dos valores y devuelve un número entero como resultado.
\$eq	compara dos valores y devuelve true si son equivalentes.
\$gt	compara dos valores y devuelve true si el primero es más grande.
\$gte	compara dos valores y devuelve true si el primero es igual o más grande.
\$lt	compara dos valores y devuelve true si el primero es menor.
\$lte	compara dos valores y devuelve true si el primero es igual o menor.
\$ne	compara dos valores y devuelve true si los valores no son equivalentes.

[Nota. Elaboración Propia](#)

Desde consola:

```
db.people.aggregate(
.. { $match: { age: { $gt: 20 } } },
.. { $project: { name: 1, age: 1 } }
.. )
```

Desde MongoDB Compass.



Figura 176: Captura de Pantalla 144

[Fuente .- Elaboración Propia](#)

5.2.4. Booleanos

Un operador booleano recibe parámetros booleanos y devuelve otro booleano como resultado. Los operadores booleanos son 3 en MongoDB.

Tabla 8
Operadores Booleanos

Operador	Descripción
\$and	devuelve true si todos los parámetros de entrada son true.
\$or	devuelve true si alguno de los parámetros de entrada es true.
\$not	devuelve el valor contrario al parámetro de entrada. Si el parámetro es true, devuelve false. Si el parámetro es false, devuelve true.

[Nota. Elaboración propia](#)

Desde consola:

```
db.people.aggregate(
.. { $match: { $and: [ { isActive: true },
..                      { age: { $gte: 25 } }
..                    ]
..      }
.. )
```

Figura 177: Captura de Pantalla 145

Fuente .- Elaboración Propia

Desde MongoDB Compass:

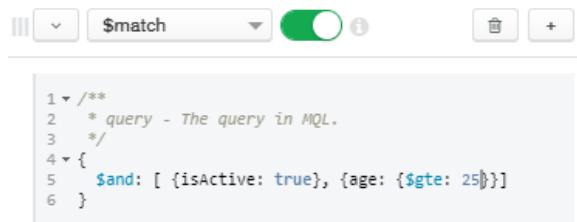


Figura 178: Captura de Pantalla 146

Fuente .- Elaboración Propia

5.2.5. Condicionales

Los operadores condicionales en **MongoDB** se utilizan para verificar una expresión y según el valor de esta expresión, devolver un resultado u otro. En **Aggregation Framework** existen dos operadores condicionales.

Tabla 9
Operadores Condicionales

Operador	Descripción
\$cond	operador ternario que recibe 3 expresiones. Si la primera es true, devuelve la segunda. Si es false, devuelve la tercera.
\$ifNull	operador que recibe 2 parámetros y en el caso de que el primero sea null, devuelve el segundo.

Nota. Elaboración propia

Desde consola:

```
db.people.aggregate(
.. { $project: { _id: 1, name: 1,
..               tipo: {$cond: [{$gte: ["$age", 65]}, "Pensionista", "Normal"]}
..             }
.. })
```

Figura 179: Captura de Pantalla 147

Fuente .- Elaboración Propia

Desde MongoDB Compass.



```

1 /**
2  * specifications - The fields to
3  * include or exclude.
4 */
5 {
6   _id: 1, name: 1,
7   tipo: {$cond: [{${gte: ["$age", 65]}, "Pensionista"}, {"$Normal"]}}
8 }
9

```

Figura 180: Captura de Pantalla 148
Fuente .- Elaboración Propia

5.2.6. Cadenas

Los operadores de cadena son aquellos que se utilizan para realizar operaciones con cadenas.

Tabla 10
Operadores de Cadena

Operador	Descripción
\$dayOfYear	concatena dos o más strings
\$dayOfMonth	crea un substring con una cantidad determinada de caracteres.
\$dayOfWeek	convierte el string a minúsculas.
\$month	convierte el string a mayúsculas.
\$week	compara dos strings y devuelve un entero con el resultado

Nota. Elaboración Propia

Desde consola:

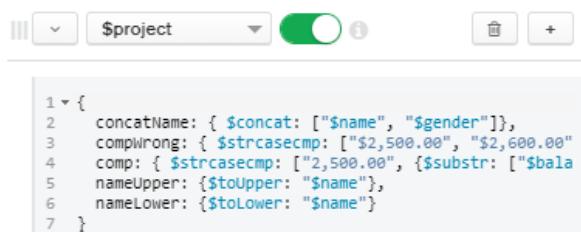
```

db.people.aggregate(
.. { $project: { concatName: { $concat: ["$name", "$gender"]},
..   compWrong: { $strcasecmp: ["$2,500.00", "$2,600.00"]},
..   comp: { $strcasecmp: ["2,500.00", {$substr: ["$balance", 1, 20]}]},
..   nameUpper: {"$toUpperCase": "$name"},
..   nameLower: {"$toLowerCase": "$name"}
.. }
.. })

```

Figura 181: Captura de Pantalla 149
Fuente .- Elaboración Propia

Desde MongoDB Compass.



```

1 {
2   concatName: { $concat: ["$name", "$gender"]},
3   compWrong: { $strcasecmp: ["$2,500.00", "$2,600.00"]
4   comp: { $strcasecmp: ["2,500.00", {$substr: ["$bal
5   nameUpper: {"$toUpperCase": "$name"}, 
6   nameLower: {"$toLowerCase": "$name"} 
7 }

```

Figura 182: Captura de Pantalla 150
Fuente .- Elaboración Propia

El uso de estos operadores es sencillo. Existen 2 tipos, los que reciben un solo parámetro y los que reciben un array con todos los parámetros necesarios. Los que reciben un solo parámetro son **\$toUpperCase** y **\$toLowerCase**, que simplemente convierten el String que llega como parámetro a mayúsculas o minúsculas. De los que reciben un array, **\$concat** simplemente concatena todos los scripts del array en uno solo. También, se encuentra **\$strcasecmp** que compara dos strings devolviendo un entero como resultado. Finalmente, **\$substr** que suele funcionar este comando en cualquier lenguaje de programación para extraer una parte de la cadena.

```
{
    "_id" : ObjectId("5dbaef73b1a239638b0c15ee"),
    "concatName" : "Maddox Daughertymale",
    "complWrong" : 0,
    "comp" : 1,
    "nameUpper" : "MADDOX DAUGHERTY",
    "nameLower" : "maddox daugherty"
}
{
    "_id" : ObjectId("5dbaef73b1a239638b0c15ef"),
    "concatName" : "Vilma Guerrafemale",
    "complWrong" : 0,
    "comp" : -1,
    "nameUpper" : "VILMA GUERRA",
    "nameLower" : "vilma guerra"
}
```

Figura 183: Captura de Pantalla 151

Fuente .- Elaboración Propia

5.2.7. Fechas

Los operadores de fecha se utilizan para realizar operaciones con campos de tipo fecha.

Tabla 11
Operadores de Fecha

Operador	Descripción
\$dayOfYear	convierte una fecha a un número entre 1 y 366.
\$dayOfMonth	convierte una fecha a un número entre 1 y 31.
\$dayOfWeek	convierte una fecha a un número entre 1 y 7.
\$month	convierte una fecha a un número de año.
\$week	convierte una fecha a un número entre el 1 y el 12.
\$hour	convierte una fecha a un número entre 0 y 53.
\$minute	convierte una fecha a un número entre 0 y 23.
\$second	convierte una fecha a un número entre 0 y 59.
\$millisecond	convierte una fecha a un número entre 0 y 59.
\$dayOfYear	devuelve los milisegundos de la fecha, con un número entre 0 y 999.

Nota. Elaboración Propia

Antes de ver un ejemplo, se tiene que hacer una transformación.

El conjunto de datos de prueba tiene un campo fecha llamado **registered**. El problema es que este campo es string, así que, debemos transformarlo para operarlos como fecha.

```
db.people.find().forEach(function (element) {
.. element.registered = ISODate(element.registered);
.. db.people.save(element)
.. })
```

Figura 184: Captura de Pantalla 152

Fuente .- Elaboración Propia

Hay que recordar que la consola de MongoDB funciona con un motor de JavaScript, por lo que puede hacerse uso del lenguaje para realizar operaciones. En este caso, todos los documentos en el campo registered debe ser transformado de string a **ISODate**.

```
db.people.find().forEach(function (element) {
    element.registered = ISODate(element.registered);
    db.people.save(element);
})
```

Figura 185: Captura de Pantalla 153
Fuente .- Elaboración Propia

Recordemos que la consola de MongoDB funciona con motor de JavaScript, por lo que siempre puedes hacer uso de este lenguaje para realizar operaciones. De esta manera, busca todos los documentos y transforma el campo **registered** de string a **ISODate**.

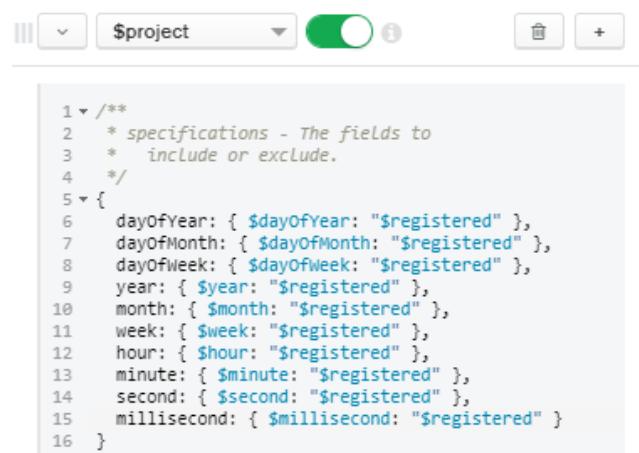
Una vez transformados los datos, puedes ejecutar una consulta de ejemplo.

Desde consola:

```
db.people2.aggregate(
.. { $project: { dayOfYear: { $dayOfYear: "$registered" },
..                 dayOfMonth: { $dayOfMonth: "$registered" },
..                 dayOfWeek: { $dayOfWeek: "$registered" },
..                 year: { $year: "$registered" },
..                 month: { $month: "$registered" },
..                 week: { $week: "$registered" },
..                 hour: { $hour: "$registered" },
..                 minute: { $minute: "$registered" },
..                 second: { $second: "$registered" },
..                 millisecond: { $millisecond: "$registered" }
..   }
.. })
```

Figura 186: Captura de Pantalla 154
Fuente .- Elaboración Propia

Desde MongoDB Compass.



```
1 /**
2  * specifications - The fields to
3  * include or exclude.
4 */
5 {
6     dayOfYear: { $dayOfYear: "$registered" },
7     dayOfMonth: { $dayOfMonth: "$registered" },
8     dayOfWeek: { $dayOfWeek: "$registered" },
9     year: { $year: "$registered" },
10    month: { $month: "$registered" },
11    week: { $week: "$registered" },
12    hour: { $hour: "$registered" },
13    minute: { $minute: "$registered" },
14    second: { $second: "$registered" },
15    millisecond: { $millisecond: "$registered" }
16 }
```

Figura 187: Captura de Pantalla 155
Fuente .- Elaboración Propia

Resumen

1. Los índices brindan un punto importante a la hora de realizar consultas contra una base de datos MongoDB. Una mala gestión de índices puede producir un bajo rendimiento.
2. Es importante
4. Sabiendo qué campos se utilizan para filtrar y cuáles suelen ser los devueltos en las consultas, es importante ser capaz de crear índices necesarios para que la base de datos se comporte correctamente.
4. El teorema CAP dice que en sistemas distribuidos (NoSQL) es imposible garantizar a la vez:
 - Consistencia
 - Disponibilidad
 - Tolerancia a particiones

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://www.youtube.com/watch?app=desktop&v=bZPZlwdu8U>
- <https://www.youtube.com/watch?app=desktop&v=CuAYLX6reXE>
- <https://www.youtube.com/watch?app=desktop&v=-0Bk3GfqRbY>
- <https://www.youtube.com/watch?app=desktop&v=0xx3fcYcb28>



OPTIMIZACIÓN DE MONGODB

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno aplica las mejores prácticas de gestión de base de datos.

TEMARIO

6.1 Tema 13 : Optimización de MongoDB

- 6.1.1 : Introducción
- 6.1.2 : Tipos de índices
- 6.2.3 : Opciones de índices
- 6.2.4 : Utilitarios de base de datos

ACTIVIDADES PROPUESTAS

- Los alumnos realizan las lecturas propuestas.
- Los alumnos realizan operaciones de consulta a MongoDB.

6.1. OPTIMIZACIÓN DE MONGODB

6.1.1. Introducción

Una parte importante en cualquier base de datos son los índices. Los índices se utilizan para incrementar la velocidad de acceso a los datos. En lugar de recorrer toda una tabla o colección para buscar un conjunto de resultados, se busca directamente en el índice que devuelve el registro o conjunto de registros de forma directa.

MongoDB define índices a nivel de colección, pudiendo reducir drásticamente el tiempo que tarda una consulta en devolver los resultados debido a que mantiene los índices en la memoria RAM, o si no es posible, en disco de forma secuencial. Este índice se almacena internamente como B-Tree (árbol-B), por lo que su creación y mantenimiento puede influir en el rendimiento de las consultas de escritura como inserciones y actualizaciones.

¿Qué es la cardinalidad?

Equivalente a un diccionario, es decir, palabras están ordenadas en orden alfabético, y son únicas. Por ello, es relativamente fácil encontrar la palabra que se busca. Ahora, imagina que el diccionario está solo agrupado y ordenado por la primera letra de cada palabra. Tendremos miles de palabras que empiezan por A, otros miles que empiezan por M, muchas otras que empiezan por la P entre otros. Entonces cuantos más valores únicos tenga el campo, más alta será la cardinalidad. Y más eficiente será el índice. (El Amigo Informático, 2020).

MongoDB accede a los documentos de dos maneras:

- **Recorriendo las colecciones;** comenzando desde el principio y extrayendo los documentos que cumplen las condiciones de la consulta; lo cual implica posicionar las cabezas lectoras, leer el dato, controlar si coincide con lo que se busca.
- **Empleando índices;** recorriendo la estructura de árbol del índice para localizar los documentos y extrayendo los que cumplen las condiciones de la consulta (comparando con un libro, es como leer el índice y luego de encontrar el tema buscado, ir directamente a la página indicada).

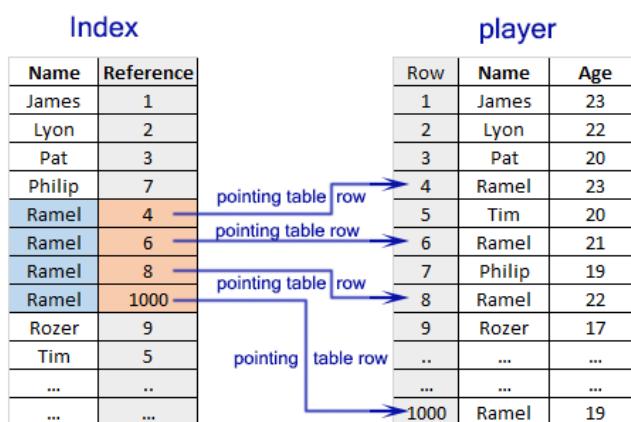


Figura 188: MongoDB Index

Fuente. - Tomado de <https://www.w3resource.com/mongodb/introduction-to-mongodb-index.php>

6.1.2. Tipos de índices

Para crear un nuevo índice en la colección emplea el método ***createIndex()***:

Sintaxis:

```
db.<collection>.createIndex(<key and index type specification>, <options>)
```

Donde:

- **Field** nombre del campo del documento.
- **Key** representa el documento con los campos de la colección a indexar.
- **Type** tipo de índice a crear.
- **Options** indicar los valores de la creación del índice.

Single Field Index

Los índices simples permiten definir un índice sobre un único campo. El orden en este caso (ascendente o descendente) no es relevante porque MongoDB puede ir en ambas direcciones.

```
db.collection.createIndex ({ campo1: 1 });
```

```
> db.maelibros.createIndex({titulo : 1})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

Figura 189: Captura de Pantalla 156
Fuente .- Elaboración Propia

Nota: el segundo parámetro es el orden (1 = ascendente y 2 = descendente)

Compound Index

Permite establecer un índice sobre varios atributos de un documento. En este caso, el orden de los atributos es relevante porque el índice será ordenado según el orden especificado.

```
db.collection.createIndex ({ campo1: 1, campo2: -1 })
```

```
> db.maelibros.createIndex({titulo : 1, autor : 1})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 3,
    "numIndexesAfter" : 4,
    "ok" : 1
}
```

Figura 190: Captura de Pantalla 157
Fuente .- Elaboración Propia

Nota: el segundo parámetro es el orden (1 = ascendente y 2 = descendente)

Multicolumn Index

Permite la indexación del contenido de un array. Si indexas un atributo de un array, se creará automáticamente entradas separadas para cada elemento del array en la tabla de índices, esto lo hace eficiente para las búsquedas de valores por campo de arrays.

```
db.collection.createIndex( { <field1>: <type>, <field2>: <type2>, ... } )
```

```
> db.prueba.createIndex({ "tags.text":1})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

Figura 191: Captura de Pantalla 158

Fuente .- Elaboración Propia

Nota: no puede crearse índices multicolumna en 2 campos de tipo array del mismo índice.

Subdocument Index

Podemos crear índices que abarquen todo el contenido de un subdocumento.

```
db.maeRepartidor.insertOne({ id: 1, apellidos: "PEREZ LOPEZ",
    nombres: "JUAN JOSE",
    direccion: {calle: "AV. GREGORIO SANCHEZ 351",
        departamento: "LIMA", provincia: "LIMA",
        distrito: "CERCADO DE LIMA"}
})
db.maeRepartidor.createIndex({direccion:1})
db.maeRepartidor.find({"direccion.departamento": "LIMA",
    "direccion.provincia": "LIMA"})
```

Figura 192: Captura de Pantalla 159

Fuente .- Elaboración Propia

Nota: MongoDB crea por cada un índice por cada atributo del campo dirección.

Embedded Index

Podemos crear índices dentro de un campo específico de un subdocumento.

```
db.maeRepartidor.insertOne({ id: 1, apellidos: "PEREZ LOPEZ",
    nombres: "JUAN JOSE",
    direccion: {calle: "AV. GREGORIO SANCHEZ 351",
        departamento: "LIMA", provincia: "LIMA",
        distrito: "CERCADO DE LIMA"}
})
db.maeRepartidor.createIndex({ "direccion.calle": 1 })
db.maeRepartidor.find({"direccion.calle": "AV. GREGORIO SANCHEZ 351"})
```

Figura 193: Captura de Pantalla 160

Fuente .- Elaboración Propia

Nota: este tipo de índice es para uso muy particular de búsqueda.

6.1.3. Opciones de índices

Unique

Es un índice que no acepta valores duplicados para el atributo indexado dentro de una colección.

Desde consola:

```
> db.clientes.createIndex({dni : 1}, { unique : true })
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

Figura 194: Captura de Pantalla 161

Fuente .- Elaboración Propia

Sparse

Una propiedad sparse en un índice hace que el índice solo contenga entradas para documentos que tienen el campo indexado (si el documento no tiene el campo que estamos indexando, no estará indexado)..

Desde consola:

```
db.clientes.createIndex({dni : 1}, { sparse : true })
```

Figura 195: Captura de Pantalla 162

Fuente .- Elaboración Propia

Background

Al crear un índice se bloquea la colección. Si esta colección tiene una gran cantidad de datos el sistema puede verse afectado por largo tiempo. Para evitar esto, el índice background permite seguir utilizando la colección mientras termina el proceso.

Desde consola:

```
db.clientes.createIndex({dni : 1}, {background : true})
```

Figura 196: Captura de Pantalla 163

Fuente .- Elaboración Propia

Borrar duplicados

Si intentas crear un índice único sobre un campo que ya contiene valores duplicados, MongoDB devolverá un error. Para evitarlo la opción dropDups borrará los duplicados.

Desde consola:

```
db.clientes.createIndex({dni : 1}, {unique:true, dropDups: true})
```

Figura 197: Captura de Pantalla 164
Fuente .- Elaboración Propia

6.1.4. Utilitarios de la base de datos

Verificación de la versión

El comando para verificar la versión instalada del servidor MongoDB y Mongo Shell.

```
mongod --version
```

```
C:\Windows\System32>mongod --version
db version v4.2.7
git version: 51d9fe12b5d19720e72dcd7db0f2f17dd9a19212
allocator: tcmalloc
modules: none
build environment:
    distmod: 2012plus
    distarch: x86_64
    target_arch: x86_64
```

Listado de comandos MongoDB

Mongo Shell retornará una lista de comando. A continuación, se lista los principales.

Tabla 12
Lista de Comando

Operador	Descripción
db.auth	Permite a un usuario autenticarse en la base de datos desde dentro del Shell. Debe estar conectado con permiso de Admin.
db.cloneDatabase	Copia una base de datos remota a la base de datos actual. El comando asume que la base de datos remota tiene el mismo nombre que la base de datos actual. Debe estar conectado con permiso de Admin. (Kroenke, 2003).
db.copyDatabase	Copia una base de datos de una instancia de MongoDB a la instancia de MongoDB actual. Cabe señalar que debe tener los permisos de Admin.
db.createCollection	Crea una nueva colección o vista. Debe estar conectado con permiso de Admin.
db.createUser	Crea un nuevo usuario en la base de datos. Devuelve un error de usuario duplicado, si el usuario existe. Debe estar conectado con permiso de Admin.
db.dropDatabase	Elimina la base de datos actual, eliminando los archivos de datos asociados. Debe estar conectado con permiso de Admin.
db.dropUser	Elimina un usuario de base de datos, eliminando los objetos

	que estén asociados. Debe estar conectado con permiso de Admin.
db.fsyncLock	Obliga al MongoDB a descargar todas las operaciones de escritura pendientes en el disco y bloquea toda la instancia de MongoDB para evitar escrituras adicionales hasta que el usuario libere el bloqueo con el comando db.fsyncUnlock. Debe estar conectado con permiso de Admin.
db.fsyncUnlock	Reduce el bloqueo tomado por db.fsyncLock en una instancia MongoDB. Debe estar conectado con permiso de Admin.
db.getCollection	Devuelve una colección o un objeto de vista. Permite devolver el nombre cuyos nombres comienzan con _ o que coinciden con un método de shell de base de datos.
db.getCollectionInfos	Devuelve una matriz de documentos con información de la colección o vista. El resultado depende de los privilegios que tiene el usuario.
db.getCollectionNames	Devuelve una matriz que contiene los nombres de todas las colecciones y vistas en la base de datos actual. El resultado depende de los privilegios que tiene el usuario.
db.hostInfo	Devuelve un documento con información del sistema en el que se ejecuta MongoDB. Debe estar conectado con permiso de Admin.
db.killOp	Termina una operación según el ID de operación. Debe estar conectado con permiso de Admin.
db.logout	Desconectarse.
db.printCollectionStats	Devuelve estadísticas de cada colección separadas por guiones.
db.serverStatus	Devuelve un documento que proporciona una descripción general del estado del proceso de la base de datos.
db.shutdownServer	Cierra el proceso actual de MongoDB de forma limpia y segura. Debe estar conectado con permiso de Admin.

Nota. Elaboración Propia

Estadísticas de base de datos

El comando para verificar el status de la base de datos desde MongoDB.

```
> db.stats()
{
    "db" : "test",
    "collections" : 0,
    "views" : 0,
    "objects" : 0,
    "avgObjSize" : 0,
    "dataSize" : 0,
    "storageSize" : 0,
    "numExtents" : 0,
    "indexes" : 0,
    "indexSize" : 0,
    "scaleFactor" : 1,
    "fileSize" : 0,
    "fsUsedSize" : 0,
    "fsTotalSize" : 0,
    "ok" : 1
}
```

Utilidades de base de datos

Este comando permite la creación de una nueva base de datos si no existe o ayuda a cambiar una base de datos existente. En MongoDB, "**prueba**" es la base de datos predeterminada para iniciar las operaciones una vez iniciada la sesión en Mongo Shell.

show dbs

Permite listar todas las bases de datos.

```
show dbs

mongo> show dbs
admin          0.000GB
config         0.002GB
geekFlareDB   0.000GB
test           0.000GB
```

db

Permite ver la base de datos actualmente en uso.

```
show dbs
```

```
Test
```

db.dropDatabase

Elimina una base de datos. Validar la selección de la base de datos antes de ejecutar el comando, de lo contrario, se eliminará la base de datos "**prueba**".

```
> show dbs (primero listar las bases de datos)
```

```

admin      0.000GB
config     0.001GB
local      0.000GB
test       0.000GB
training   0.000GB
>

> use training
switched to db training (activo la base de datos a eleminar)
>

> db.dropDatabase()
{ "dropped" : "training", "ok" : 1 } (elimino la base de datos)

```

db.mongodump

Exportar el contenido de la base de datos MongoDB como copia de seguridad. Este comando se ejecuta desde la consola del sistema y no desde mongo shell. Generará una copia de seguridad binaria junto con la información de metadatos

```
mongodump --db dbName
--out outFile --host "IP:PORT" --username <user> --password <pass>
```

Ejemplo con la base de datos db geekFlareDB

```

mongodump --db geekFlareDB --out "C:\mongodb\dump" --host "127.0.0.1:27017"
2020-06-02T12:26:34.428+0530  writing geekFlareDB.myTable to
2020-06-02T12:26:34.430+0530  writing geekFlareDB.geekFlareCollection to
2020-06-02T12:26:34.430+0530  writing geekFlareDB.mCollection to
2020-06-02T12:26:34.433+0530  writing geekFlareDB.users to
2020-06-02T12:26:34.434+0530  done dumping geekFlareDB.myTable (2 documents)
2020-06-02T12:26:34.434+0530  done dumping geekFlareDB.geekFlareCollection
2020-06-02T12:26:34.435+0530  writing geekFlareDB.contacts2 to
2020-06-02T12:26:34.435+0530  writing geekFlareDB.Login to
2020-06-02T12:26:34.436+0530  done dumping geekFlareDB.mCollection
2020-06-02T12:26:34.437+0530  done dumping geekFlareDB.users (1 document)
2020-06-02T12:26:34.437+0530  done dumping geekFlareDB.Login (0 documents)
2020-06-02T12:26:34.438+0530  done dumping geekFlareDB.contacts2 (0 documents)

```

db.mongorestore

Se utiliza para restaurar datos binarios generados por mongodump.

```

mongorestore --db geekFlareNew
2020-06-09T15:49:35.147+0530  the --db and --collection args should only be used
when restoring from a BSON file. Other uses are deprecated and will not exist in the
future; use --nsInclude instead
2020-06-09T15:49:35.148+0530  building a list of collections to restore from
C:\mongodb\dump\geekFlare dir
2020-06-09T15:49:35.152+0530  reading metadata for
geekFlareNew.geekFlareCollection from
C:\mongodb\dump\geekFlare\geekFlareCollection.metadata.json

```

```
2020-06-09T15:49:35.321+0530  restoring geekFlareNew.geekFlareCollection from
C:\mongodb\dump\geekFlare\geekFlareCollection.bson
2020-06-09T15:49:35.461+0530  no indexes to restore
2020-06-09T15:49:35.462+0530  finished restoring geekFlareNew.geekFlareCollection
(3 documents, 0 failures)
2020-06-09T15:49:35.467+0530  3 document(s) restored successfully. 0 document(s)
failed to restore.
```

Explain ()

El método devuelve estadísticas y proporciona información para seleccionar un plan de ejecución con mejor performance. Devuelve resultados según el plan de verbosidad.

```
collectionName.explain("verbosityName")
```

Ejemplo con la base de datos db geekFlareDB

```
> db.runCommand(
...  {
...    explain: { count: "product", query: { Qty: { $gt: 10 } } },
...    verbosity: "executionStats"
...  }
...
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "test.product",
    "indexFilterSet" : false,
    "winningPlan" : {
      "stage" : "COUNT",
      "inputStage" : {
        "stage" : "EOF"
      }
    },
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 0,
    "executionTimeMillis" : 47,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 0,
    "executionStages" : {
      "stage" : "COUNT",
      "nReturned" : 0,
      "executionTimeMillisEstimate" : 0,
      "works" : 1,
      "advanced" : 0,
      "needTime" : 0,
      "needYield" : 0,
      "saveState" : 0,
      "restoreState" : 0,
    }
  }
}
```

```

    "isEOF" : 1,
    "nCounted" : 0,
    "nSkipped" : 0,
    "inputStage" : {
        "stage" : "EOF",
        "nReturned" : 0,
        "executionTimeMillisEstimate" : 0,
        "works" : 0,
        "advanced" : 0,
        "needTime" : 0,
        "needYield" : 0,
        "saveState" : 0,
        "restoreState" : 0,
        "isEOF" : 1
    }
},
"serverInfo" : {
    "host" : "MCGL-4499",
    "port" : 27017,
    "version" : "4.2.7",
    "gitVersion" : "51d9fe12b5d19720e72dcd7db0f2f17dd9a19212"
},
"ok" : 1
}
>

```

Transacciones en MongoDB

MongoDB admite propiedades ACID para transacciones en documentos. Cabe señalar que este concepto lo hemos revisado en el capítulo 2. Para iniciar una transacción, se requiere una sesión para comenzar y se requiere un compromiso para guardar los cambios en la base de datos. Las transacciones se admiten en réplicas. Una vez que la sesión se comprometió con éxito, las operaciones realizadas dentro de la sesión serán visibles en el exterior.

- Iniciar sesión: session = db.getMongo().startSession()
- Iniciar transacción: session.startTransaction()
- Confirmar transacción: session.commitTransaction()

Ejemplo con la base de datos db geekFlareDB

Creemos una sesión, iniciemos la transacción. Luego, realizamos la inserción y/o actualización y finalmente confirmamos la transacción. El proceso es igual que en las bases de datos relacionales.

```

rs0:PRIMARY> session = db.getMongo().startSession()
session { "id" : UUID("f255a40d-81bd-49e7-b96c-9f1083cb4a29") }
rs0:PRIMARY> session.startTransaction()
rs0:PRIMARY> session.getDatabase("geekFlareDB").geekFlareCollection.insert([
... { _id: 4 , product: "ketchup"},
```

```
... {_id: 5, product: "Cheese"}  
... ]);  
BulkWriteResult({  
    "writeErrors" : [ ],  
    "writeConcernErrors" : [ ],  
    "nInserted" : 2,  
    "nUpserted" : 0,  
    "nMatched" : 0,  
    "nModified" : 0,  
    "nRemoved" : 0,  
    "upserted" : [ ]  
})  
rs0:PRIMARY> session.getDatabase("geekFlareDB").geekFlareCollection.find()  
{ "_id" : 1, "product" : "bread", "Qty" : 20 }  
{ "_id" : 2, "product" : "bottles", "Qty" : 100 }  
{ "_id" : 3, "product" : "bread", "Qty" : 20 }  
{ "_id" : 4, "product" : "ketchup" }  
{ "_id" : 5, "product" : "Cheese" }  
rs0:PRIMARY> db.geekFlareCollection.find()  
{ "_id" : 1, "product" : "bread", "Qty" : 20 }  
{ "_id" : 2, "product" : "bottles", "Qty" : 100 }  
{ "_id" : 3, "product" : "bread", "Qty" : 20 }  
rs0:PRIMARY> session.commitTransaction()  
rs0:PRIMARY> db.geekFlareCollection.find()  
{ "_id" : 1, "product" : "bread", "Qty" : 20 }  
{ "_id" : 2, "product" : "bottles", "Qty" : 100 }  
{ "_id" : 3, "product" : "bread", "Qty" : 20 }  
{ "_id" : 4, "product" : "ketchup" }  
{ "_id" : 5, "product" : "Cheese" }  
rs0:PRIMARY> session.getDatabase("geekFlareDB").geekFlareCollection.find()  
{ "_id" : 1, "product" : "bread", "Qty" : 20 }  
{ "_id" : 2, "product" : "bottles", "Qty" : 100 }  
{ "_id" : 3, "product" : "bread", "Qty" : 20 }  
{ "_id" : 4, "product" : "ketchup" }  
{ "_id" : 5, "product" : "Cheese" }
```

Resumen

1. Los índices son un punto importante a la hora de realizar consultas en una base de datos MongoDB. Una mala gestión de índices puede producir un bajo rendimiento.
2. Antes de crear índices debemos tener claro los campos que son utilizados en las consultas. En consecuencia, los desarrolladores y administradores tienen un rol importante para mantener los buenos rendimientos de la base datos.
3. A nivel de gestor de bases datos debemos considerar un mantenimiento y respaldo constante para garantizar la continuidad del servicio ante un evento negativo.

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://www.youtube.com/watch?v=K2Ib9DM-5qg>
- <https://www.youtube.com/watch?v=DJ5Wnkn5w0g>
- <https://www.youtube.com/watch?v=NbAFz5rxoDs>
- <https://www.youtube.com/watch?v=bdS03tgD2QQ>

Bibliografía

- Alonso Vega, Adrian (2021) *5 razones para probar MongoDB*. Recuperado de <https://adrianalonso.es/desarrollo-web/nosql/5-razones-para-probar-mongodb/>
- Bradshaw, Shannon; Brazil, Eoin & Chodorow, Kristina (2019) *MongoDB: The Definitive Guide*. 3rd ed. Sebastopol, California: O'Reilly Media, Inc.
- Caballero, Rafael (2017) *Bases de datos NoSQL*. Recuperado de <http://gpd.sip.ucm.es/rafa/docencia/nosql/index.html>
- Charte Ojeda, Francisco (2015) *Manual imprescindible de SQL*. Madrid: Anaya Multimedia.
Centro de Información: Código 005.756 CHAR/M
- CodeHoven (2019) *MongoDB: Aggregation*. Recuperado de <https://www.codehoven.com/mongodb-aggregate-operaciones-agregacion-pipeline/>
- CodeHoven (2018) *MongoDB 101: CRUD operations*. Recuperado de <https://www.codehoven.com/mongodb-update-insert-find-delete/>
- El Amigo Informático (2020) *Índices en MongoDB*. Recuperado de <http://elamigoinformatico.org/programacion/indices-en-mongodb/>
- Faci, Santiago (2019) *Acceso a Bases de Datos NoSQL*. MongoDB. Recuperado de <https://datos.codeandcoke.com/apuntes:mongodb>
- Homeworkdatabase (2015) *SGBD o SMBD: "Sistema Manejador de Base de Datos"*. Recuperado de <https://homeworkdatabase.wordpress.com/2015/06/27/smbd-o-smbd-sistema-manejador-de-base-de-datos/>
- Kroenke, David (2003) *Procesamiento de bases de datos: fundamentos, diseño e implementación*. 8a ed. México, D.F.: Pearson Educación.
Centro de Información: Código 005.74 KROE 2003
- Moisset, Diego (s.f.) *MongoDB*. Recuperado de <https://www.tutorialesprogramacionya.com/mongodbya/>
- MongoDB (2021) *The MongoDB 5.0 Manual*. Recuperado de <https://docs.mongodb.com/manual/>
- MongoDB Inc (2021) *Welcome to the MongoDB Documentation*. Recuperado de <https://docs.mongodb.com/>
- Phaltankar, A.; Ahsan, J.; Harrison, M.; Nedov L. (2020) *MongoDB Fundamentals: A hands-on guide to using MongoDB & Atlas in the real world*. USA: Packt Publishing Ltd.
- Robledano, Angel (2019) *Qué es MongoDB*. Recuperado de <https://openwebinars.net/blog/que-es-mongodb/>

- Saiz, José Antonio (2014) *MongoDB para Principiantes*. Recuperado de <http://joseantoniossaiz.com/mongodb-para-principiantes/>
- Sarasa, Antonio (2016) *Introducción a las bases de datos NOSQL usando MongoDB* (EBOOK). Barcelona: UOC (UNIVERSITAT OBERTA DE CATALUNYA).
- Silberschatz, Abraham (2002) *Fundamentos de bases de datos*. 4a ed. Madrid: McGraw-Hill.
Centro de Información: Código 005.74 SILB 2002
- Tecnologias-Informacion.com (2018) *Bases de Datos Clave Valor*. Recuperado de <https://www.tecnologias-informacion.com/clave-valor.html>
- Watkins, A. (2019) *MongoDB: Aprendamos a usar bases de datos NoSQL*. USA: Kindle.