**Unidad 1**

**MongoDB**

**MongoDB** es un sistema manejador de base de datos **NoSQL** orientado a documentos, por lo cual las bases de datos se componen de **colecciones** que incluyen **documentos**. La creación de una base de datos consiste así en crear colecciones y dentro de cada colección documentos, los cuales son equivalentes a las tablas en las bases de datos SQL. Sin embargo, los documentos no son estructurados como las tablas, permitiendo así tener una variedad de elementos dentro de los documentos que componen una colección, que ofrece una mayor flexibilidad a la hora de diseñar la base de datos.

Al igual que en MySQL y Postgres, MongoDB ofrece mecanismos de **recuperación** y **respaldo** de la base de datos que operan de forma muy similar. También, como en los otros manejadores relacionales, MongoDB tiene opciones para poder **importar** los datos y **exportar** los mismos.

MongoDB proporciona diversas formas de consultar los documentos en las colecciones incluidas en nuestra base de datos con la función **find**, a través de la cual podemos añadir otras funciones para crear consultas más complejas y avanzadas.

Otra forma de consultar, llamada **consultas agregadas**, se construye usando una estructura de **tuberías** (o pipelines) de diferentes etapas, donde una salida es la entrada de otra. Los elementos de la tubería se colocan en un **arreglo** y se ejecutan por orden. Otros de los elementos que hemos estudiado en la unidad son el **$project** que permite incluir o excluir campos, y el **$match** para filtrar documentos.

Finalmente, cerramos la unidad con las operaciones de autenticación para definir usuarios y sus roles en el sistema. Los roles permiten limitar el acceso de estos a la base de datos. MongoDB tiene roles predefinidos pero permite a su vez la creación de otros.

**Unidad 2**

**CouchDB**

**CouchDB** es un manejador de base de datos **NoSQL** orientado a documentos Json caracterizado por: el uso de **API REST** para hacer peticiones **HTTP**, facilitar el desarrollo de aplicaciones **Mobile First**, poseer un lenguaje propio para realizar consultas, llamado **Mango Query**, y tener un manejo dinámico de versiones de los documentos. Adicionalmente, CouchDB es **escalable** permitiendo que se agreguen más nodos al sistema de base de datos, mientras tiene un manejo de **replicaciones** de dichos nodos que facilita la sincronización de las bases de datos distribuidas; todo ello contribuye a un eficiente **manejo del tráfico** de peticiones a las base de datos.Este manejador puede ser accedido a través de un terminal de comando, como el **curl** o una Aplicación Web como **Fauxton**.

La creación de las bases de datos y documentos en CouchDB se realiza a través de peticiones conocidas en HTTP, tales como **GET**, **POST**, **PUT** y **DELETE** hacia un recurso (/), pero que tienen una sintaxis y semántica un poco diferente en el contexto de este manejador.

Las operaciones de consultas se dividen en dos. Se pueden realizar consultas a través del uso del lenguaje **Mango Query** o **find** usando diversas opciones, tales como el **selector**, **limit** y **sort**. Por el otro, se pueden crear estructuras llamadas **vistas** que son el resultado de las consultas usando una **Función Map**, que se puede codificar a través del uso del lenguaje de programación **JavaScript**. Esta función junto al uso del **reduce** permite el agrupamiento de los datos.

Las **funciones de validación**  conllevan a validar los documentos para que mantengan el formato y la estructura requeridos; mientras que las **operaciones de seguridad** para crear usuarios y sus permisos permiten definir y validar el acceso a nuestros recursos. Finalmente, la **recuperación en lote** se puede definir como una **petición POST** a un archivo que contiene los datos en formato Json; mientras que el **respaldo**, se puede hacer de varias formas, pero una de ellas es a través de la escritura de nuestros documentos en un archivo con formato Json, el cual debemos **transformar** si deseamos recuperarlo posteriormente usando el proceso de **recuperación en lote**.

**Unidad 3**

**REDIS**

Dentro de la diversidad de manejadores NoSQL tenemos los orientados a **Clave/Valor**, donde para cada **valor** se requiere una **clave** para accederlo. **Redis** es uno de los manejadores en esta categoría. Sus características principales son: los datos están en **memoria (InMemory)** y **permanecen** allí hasta que se venza su tiempo de vida o el servidor se apague por completo (**persistencia de los datos**), su arquitectura es cliente/servidor y los nodos se pueden agrupar para formar **clusters** y poder **replicarse** en la red, y las instrucciones se **manejan como textos** pudiendo el programador definir el significado de lo que contiene cada string. Estas características de Redis lo hace útil en aplicaciones donde debemos guardar datos en **caché** o donde tenemos **procesos cuyos estatus** deben guardarse en memoria.

A través de un conjunto básico de instrucciones, tales como el **SET** y el **GET**, podemos definir claves y valores para nuestros datos y luego consultarlos a través del uso de sus correspondientes claves. Siendo Redis un manejador **basado en texto**, los tipos de datos se definen a través del uso de strings, con la peculiaridad de que se pueden realizar operaciones matemáticas sobre estos tipos. El manejo de datos como strings permite, también, que el **programador defina la sintaxis y semántica de los datos** con la ayuda de caracteres especiales, tales como # y %. Adicionalmente, encontramos que existen otros tipos de datos más avanzados como las **listas**, **hash** y los **conjuntos** que expanden el abanico de opciones para el manejo de los mismos en estructuras más complejas.

Redis está pensado para el acceso **fácil** y **rápido** de los datos en memoria, por lo tanto los mecanismos de seguridad no son tan complejos. En esta unidad, vimos la **autenticación** a través del uso de **contraseñas** que determinan si se debe usar una clave o no para ejecutar una instrucción. Para ello hicimos uso del comando **CONFIG SET** con el **requirepass** y el **AUTH**.

Cerramos esta unidad estudiando los procesos de **respaldo** y **recuperación**, los cuales están muy ligados a la persistencia de los datos en memoria. A diferencia de otros manejadores, Redis **guarda** todos los datos del sistema en un **archivo predefinido** (tal como **dump.dbd**) antes de que el servidor se pare. Entonces para **respaldar** nuestros datos, lo que hacemos es copiar este archivo en otro directorio y posiblemente le asignamos un nuevo nombre. La **recuperación** simplemente consistirá en sobreescribir el archivo de respaldo predefinido por Redis con nuestro archivo, antes de iniciar el servidor nuevamente. Con esto el sistema colocará en memoria los datos que habíamos respaldado con anterioridad.

**Unidad 4**

**Otros maneadores NoSQL**

Una **base de datos orientada a grafos** incluye una serie de **nodos** relacionados entre sí a través de **relaciones** representadas por arcos dirigidos; tanto los nodos como las relaciones tienen **propiedades**. **Neo4j** es un manejador orientado a grafos escrito en Java y que posee un API REST. La realización de queries a la base de datos se realiza a través de un lenguaje llamado **Cypher**, que soporta operaciones básicas de manipulación de los nodos y relaciones, así como también operaciones avanzadas de agrupación y sobre los grafos.

Tanto las bases de datos relacionales como las no relacionales tienen sus ventajas, así como también sus desventajas; **NewSQL** representa una nueva tendencia en las bases de datos que toma lo mejor de los anteriores paradigmas. De las bases de datos relacionales, **NewSQL** adopta la **consistencia**, la **orientación a transacciones** y el uso de un **lenguaje estándar SQL** para operar sobre las bases de datos, mientras que de las bases de datos no relacionales toma la **escalabilidad**, soportada por el uso de una arquitectura de **nodos distribuidas**. **VoltDB** es un manejador que representa esta orientación y está basado en el manejo de los datos **en memoria** para proporcionar alta disponibilidad y acceso de forma rápida.

Un lenguaje de programación se conecta a un manejador de base de datos a través de un **conector**, que usualmente se instala como una **librería externa**. Este conector es el cliente dentro de la arquitectura **cliente/servidor** de nuestro sistema de base de datos. Para interactuar con una base de datos NoSQL desde una aplicación debemos **establecer una conexión**, **seleccionar la base de datos** a usar, **seleccionar la colección** (en caso que aplique), **realizar la consulta** e **iterar sobre los datos**. En el caso que necesitemos interactuar con una base de datos SQL debemos **crear la conexión** especificando con quién deseamos conectarnos, **definir la consulta**, **ejecutar la consulta** y **cerrar la conexión**.

Por su parte, **Object-Oriented Mapping (ORM)** es una librería que se agrega al conector del manejador de base de datos para crear un capa adicional, que permite realizar una correspondencia entre los objetos en un lenguaje orientado a objetos, tal como, Java o C++, y los elementos de la base de datos. ORM permite al programador abstraerse de los detalles y estructuras de la base de datos mientras opera desde un ambiente orientado a objetos.