BASES DE DATOS NoSQL

Los manejadores NoSQL se caracterizan por proporcionar diversos modelos de manejo de los datos mientras que se desligan de las estructuras de tablas de sus contrapartes relacionales. A este grupo de manejadores no relacionales pertenecen los orientados a documentos, los de clave/valor, los orientados a grafos entre otros. En este curso, hemos explorado cuatro de estos manejadores: MongoDB, CouchDB, Redis y Neo4j, que son representativos de la categoría a la cual representan.

Comenzamos con los manejadores orientados a documentos y estudiamos MongoDB y CouchDB, los cuales, a pesar de estar en el mismo grupo, tienen una forma disímil de implementar sus operaciones de acceso a los datos. Seguidamente, revisamos Redis un manejador que pertenece a la categoría de los orientados a clave/valor y donde sus datos se manejan desde la memoria para proporcionar rapidez en el acceso de los mismos.

Profundizamos un poco más en manejadores NoSQL incorporando a Neo4j, un manejador orientado a grafos, a nuestra base de conocimientos. Seguidamente exploramos las nuevas tendencias en bases de datos con el paradigma NewSQL y estudiadamos bervemente uno de los manejadores que ha implementado esta nueva filosofía, como los es VoltDB.

Con la finalidad de mostrar escenarios donde los componentes de las bases de datos se incorporan en el diseño e implementación de soluciones, analizamos los pasos generales para interactuar desde bases de datos SQL y NoSQL desde diversos lenguajes de programaciones, tales como Java, Javascript/Node, Python, entre otros. Dichos pasos fueron ilustrados a través de pequeños ejemplos, que nos servirán como esqueletos en la realización de nuestro propio código en el futuro.

Finalmente, cerramos estudiando la forma de interactuar con las bases de datos usando lenguajes orientados a objetos a través de Object-Oriented Mapping (ORM). Esta librería proporciona una capa intermedia entre el lenguaje de programación y la base de datos, que nos permite manipular nuestros datos usando objetos en vez de los elementos naturales de la base de datos que estemos usando.

Luego de haber culminado exitosamente el curso de Bases de Datos NoSQL has desarrollado la capacidad de:

- Aplicar los conceptos y operaciones de las bases de datos NoSQL orientadas a documentos a través del uso de la herramienta MongoDB y CouchDB.

- Aplicar los conceptos y operaciones de las bases de datos NoSQL orientadas a clave/valor a través del uso de la herramienta Redis.

- Interpretar los conceptos de las bases de datos NoSQL orientadas a grafos a través de ejemplos basados en Neo4j.

- Interpretar los principios de las bases de datos NewSQL como paradigma que agrupa las nuevas tendencias en el área.

- Diferenciar las diversas formas de interactuar con un manejador de base de datos desde un lenguaje de programación con la finalidad de poder desarrollar soluciones de software que utilicen base de datos.

Ahora estás listo para diseñar e implementar tus propias soluciones de bases de datos en diversos contextos. Pero primero, te dejamos un resumen de lo que hemos cubierto en las Bases de Datos NoSQL.

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 masajeadores 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, se llama **consultas agregadas**, que se construyen 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. Algunos 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**, el 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 una manejo de **replicaciones** de dicho nodos que facilita la sincronización de las base 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 resultados de 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 lograr el agrupamiento de los datos.

Las **funciones de validación** nos permiten 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 se debemos guardar datos en **caché** o donde tenemos **procesos cuyo estatus** debe guardarse en memoria.

A través de un conjunto básicos 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 **fácil** y **rápido** acceso 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 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 MANEJADORES 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, asi como tambien sus desventajas; **NewSQL** representa una nueva tendencia en las base 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 quien deseamos conectarnos, **definir la consulta**, **ejecutar**

la consulta y **cerrar la conexión.**

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.