**¿Qué diferencia hay entre base de datos no relacionales y relacionales?**

Base de datos relacionales.

Las bases de datos relacionales se caracterizan por ser una colección ordenada de registros que se organizan en un conjunto de tablas. Estas tablas se relacionan entre sí, dando lugar a una base de datos desde donde se puede acceder a los datos o volver a montarlos de muchas maneras diferentes sin tener que reorganizar las tablas de la base.

Para trabajar con esos datos se utiliza un lenguaje de Consultas Estructuradas llamado SQL Structured Query Language).

Las bases de datos relacionales se organizan a través de identificadores. De este modo, cada tabla tiene un identificador único que es el que va a establecer su relación con el resto de las tablas.

A su vez, estos identificadores hacen que sea más fácil organizar cada una de las tablas por separado.

En cuanto a los formato todo lo veo con el formato tabla.

Los principales sistemas gestores de bases de datos relacionales son: MySQL, MariaDB, SQLite, PostgreSQL, SQL Server y Oracle.

Bases de datos no relacionales

Las bases de datos no relacionales están diseñadas para modelos de datos específicos y que no necesitan ser relacionados con otros modelos. Cada tabla funciona de forma independiente y son mucho más sencillas que los modelos relacionales.

Esta sencillez de acceso y de ordenación de la base de datos hace que estén cobrando más importancia que las relacionales.

Las bases de datos no relacionales pueden tener identificador único, es decir, para identificar cada uno de los registros de la base de datos, pero este identificador no se usará (generalmente) para relacionar unos registros con otros. Como veremos, la información se organiza normalmente mediante documentos y es muy útil cuando no tenemos un esquema exacto de lo que se va a almacenar.

Con respecto a los formatos que se utilizan en las bases de datos no relacionales, podríamos decir que el formato más popular es el del documento. En muchos casos, lo que se utiliza es un objeto con una clave y un valor para que el acceso a la información sea pueda realizar de una forma sencilla.

Los principales sistemas gestores de bases de datos no relacionales son: MongoDB, Redis y Cassandra.

¿Cuándo utilizar SQL o NOSQL?

Después de esta explicación, podemos decir que ambos tipos de bases de datos son útiles y dependerá del tipo de aplicación que queramos realizar la elección de una u otra base de datos.

Así, si queremos desarrollar una aplicación de tipo contable, de inventario o de información de clientes, es probable que el modelo relacional se adapte mejor a nuestras necesidades. En este tipo de aplicaciones, normalmente habrá más de una tabla que tenga relación con el resto, por lo que una base de datos relacional será más útil y podrá representar mejor nuestra aplicación.

Si por el contrario nuestra aplicación necesita de un sistema en el que los registros que vaya a almacenar no necesitan relacionarse con otros, usaremos una base de datos no relacional. Estas serán muy útiles para guardar datos que sean de un solo tipo de esquema, como por ejemplo puede ser una base de datos en la que solo queramos almacenar libros o películas, ya que almacenaremos cada documento y no lo relacionaremos con ningún otro dato, solamente le daremos una serie de atributos que se podrán consultar pero que no tendrán relación alguna con otra tabla.

Por todo lo dicho anteriormente, queda claro que la elección del tipo de base de datos es algo muy importante y que no hay que tomarlo a la ligera. La elección de una base de datos errónea para el proyecto que queramos desarrollar puede traer consecuencias fatales para el desarrollo del mismo, por lo que es muy importante dedicarle tiempo y ver qué tipo de base de datos encajará mejor con nuestra aplicación.

**¿Qué diferencia hay entre JWT y Cookies, que ventajas de cada uno?**

Las cookies y sesiones para gestionar la autenticación de usuarios en mis aplicaciones web.

El JWT es sólo un formato de token, y el segundo es un mecanismo de gestión de estado para las peticiones HTTP.

Es correcto sin embargo que ambos son ampliamente utilizados en sistemas de autenticación.

La autenticación basada en cookies es conveniente para los navegadores, pero más allá de ellos, para otros tipos de clientes tiene más sentido un enfoque basado en tokens (ya que estos pueden ser transportados a través de parámetros, encabezados o como parte del cuerpo de las peticiones HTTP).

Es decir, si una API admite tokens entonces aumentará el rango de clientes que puede atender, por lo que resulta más conveniente si la API se debe usar más allá de los navegadores web.

**Ventajas**

Sin estado, escalable y desacoplado

Probablemente la mayor ventaja de usar tokens y no cookies es el hecho de que ofrecen una autenticación sin estado (stateless).

Desde backend no se necesita tener un registro de los tokens. Cada token es autónomo: contienen en sí mismos toda la data necesaria para confirmar su validez (así como también información puntual del usuario que ha iniciado sesión).

De esta forma, el único trabajo del servidor es: firmar tokens ante un inicio de sesión exitoso, y verificar que los tokens entrantes sean válidos.

**Cross Domain y CORS**

Las cookies funcionan bien con un dominio (o subdominio) en específico, pero cuando se trata de administrar cookies en diferentes dominios, el manejo se torna complicado.

Guardar datos en los JWT

Con un enfoque basado en cookies, simplemente guardamos el identificador de sesión.

Los tokens por otro lado nos permiten guardar cualquier tipo de metadata, siempre que se trate de un JSON válido.

Al utilizar una autenticación basada en cookies, desde backend se debe realizar una búsqueda de la sesión (correspondiente al identificador enviado por el cliente; ya sea en archivos, en una base de datos SQL tradicional o una alternativa NoSQL). En ese caso es muy probable que la ida y vuelta tome más tiempo si lo comparamos con la decodificación de un token. Además, como se pueden almacenar datos adicionales en los tokens (como el nivel de permisos), podemos disminuir la cantidad de búsquedas requeridas para obtener y procesar los datos solicitados.

Por ejemplo, supongamos que tenemos un recurso /api/orders en nuestra API que devuelve las últimas órdenes registradas en nuestra aplicación, pero sólo los usuarios con rol administrador tienen acceso para ver esta data.

En un enfoque basado en cookies, una vez que se realiza la petición, desde backend es necesario hacer una consulta para verificar que la sesión es válida, otra búsqueda para acceder a los datos del usuario y verificar que tenga el rol de administrador, y finalmente una tercera consulta para obtener los datos.

Por otro lado, usando JWT, podemos guardar el rol del usuario en el token. Así, una vez que la petición se realiza y el token se valida, necesitamos realizar una sola consulta a la base de datos (para acceder a la información de las órdenes).

Los tokens, por otro lado, son mucho más fáciles de implementar (tanto en iOS como en Android). También son más fáciles de implementar para aplicaciones y servicios de Internet of Things (que no incorporan el concepto de gestión de cookies)

¿Para qué sirve el protocolo OAuth?

Pertenece al tema de autenticación.

Oauth nos permite loguearnos con redes sociales, Facebook, twiter a ciertas páginas.

OAuth es un estándar abierto que permite la autorización segura mediante el uso de un API.

Oauth proporciona flujo de autorización para aplicaciones web, escritorio y teléfonos.