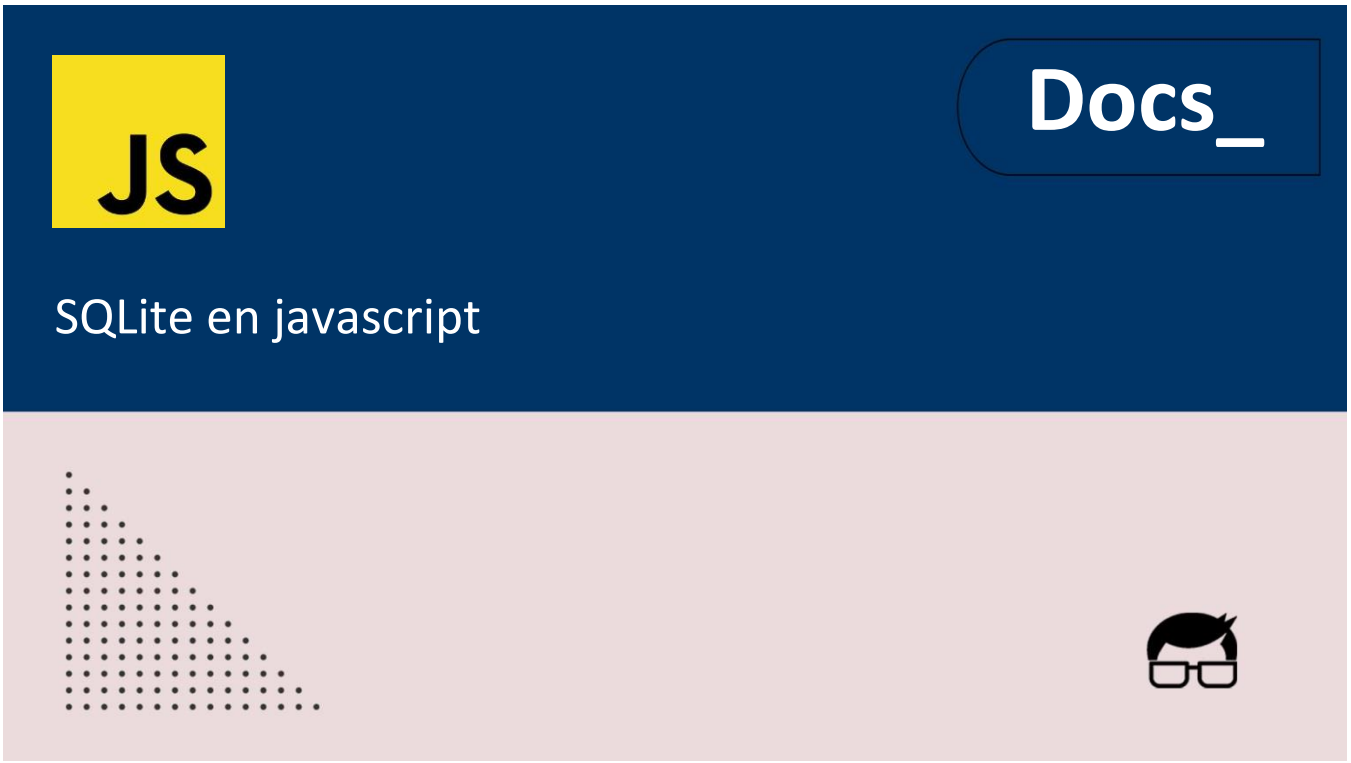

Introducción: Implementación de SQLite en un proyecto Node.js

SQLite es una biblioteca de software que proporciona un motor de base de datos SQL ligero, rápido y autocontenido. Es perfecto para aplicaciones que requieren un almacenamiento de datos simple y eficiente sin necesidad de configuraciones complicadas de servidor.

En este tutorial, exploraremos cómo integrar SQLite en un proyecto Node.js. Node.js es un entorno de ejecución de JavaScript del lado del servidor que nos permite construir aplicaciones web y servicios con facilidad y rapidez.

A lo largo de este tutorial, aprenderemos paso a paso cómo:

1. Configurar y crear una base de datos SQLite.
2. Instalar y configurar un controlador de SQLite en un proyecto Node.js.
3. Conectar nuestra aplicación Node.js a la base de datos SQLite.
4. Escribir y ejecutar consultas SQL en nuestro proyecto Node.js utilizando SQLite.
5. Manejar los resultados de las consultas y la lógica de negocio.
6. Probar y depurar nuestra aplicación para garantizar su correcto funcionamiento.

Ya sea que estés desarrollando una aplicación web, un servicio de backend o cualquier otro proyecto Node.js que requiera un almacenamiento de datos local, este tutorial te proporcionará los conocimientos necesarios para integrar SQLite de manera efectiva en tu aplicación.

@mihifidem 01/2024

Docs_ Generalidades Curiosidades

1. Configuración y creación de una base de datos SQLite

SQLite es una base de datos de un solo archivo, lo que significa que toda la base de datos se almacena en un único archivo de disco. Esto simplifica la configuración y no requiere un servidor de base de datos separado. A continuación, te mostraré cómo configurar y crear una base de datos SQLite en tu proyecto Node.js:

- SQLite es una base de datos de un solo archivo, lo que significa que toda la base de datos se almacena en un único archivo de disco.
- Para crear una base de datos SQLite, simplemente necesitas especificar la ruta y el nombre del archivo de la base de datos. Puedes hacerlo utilizando el módulo **sqlite3** en Node.js.
- Después de especificar la ruta y el nombre del archivo de la base de datos, puedes conectar tu aplicación Node.js a esta base de datos y empezar a ejecutar consultas SQL para crear tablas y manipular los datos.

Instalación del módulo SQLite: En primer lugar, necesitarás instalar el módulo **sqlite3** en tu proyecto Node.js.

```
npm install sqlite3
```

Creación de la base de datos: Ahora, puedes crear una nueva base de datos SQLite utilizando Node.js. Por ejemplo, crea un archivo llamado **database.js** en tu proyecto y escribe el siguiente código para crear la base de datos y una tabla de ejemplo:

```
const sqlite3 = require('sqlite3').verbose();
const path = require('path');

// Ruta del archivo de la base de datos
const dbPath = path.resolve(__dirname, 'my_database.db');

// Crear una nueva instancia de base de datos en la ruta especificada
const db = new sqlite3.Database(dbPath);

// Crear una tabla de ejemplo llamada 'usuarios'
db.serialize(() => {
  db.run("CREATE TABLE usuarios (id INTEGER PRIMARY KEY, nombre TEXT)");
});

// Cerrar la conexión a la base de datos cuando termine
db.close((err) => {
  if (err) {
    return console.error(err.message);
  }
  console.log('Conexión a la base de datos SQLite cerrada');
});
```

Docs_ Generalidades Curiosidades

1. **Instalar y configurar un controlador de SQLite en un proyecto Node.js:**
    - Para interactuar con una base de datos SQLite desde Node.js, necesitas instalar el módulo **sqlite3** utilizando npm.
    - Una vez instalado, puedes importar el módulo **sqlite3** en tu aplicación Node.js para utilizarlo y configurar una conexión a la base de datos SQLite.
2. **Conectar nuestra aplicación Node.js a la base de datos SQLite:**
    - Después de instalar y configurar el módulo **sqlite3**, puedes crear una instancia de la clase **Database** proporcionada por el módulo para establecer una conexión a la base de datos SQLite.
    - Esta conexión te permitirá ejecutar consultas SQL y realizar operaciones de lectura y escritura en la base de datos desde tu aplicación Node.js.
3. **Escribir y ejecutar consultas SQL en nuestro proyecto Node.js utilizando SQLite:**
    - Una vez que hayas establecido la conexión a la base de datos SQLite, puedes utilizar métodos proporcionados por el módulo **sqlite3** para ejecutar consultas SQL en la base de datos.
    - Estas consultas pueden incluir operaciones de creación, lectura, actualización y eliminación (CRUD) de datos en las tablas de la base de datos.
4. **Manejar los resultados de las consultas y la lógica de negocio:**
    - Después de ejecutar consultas SQL en la base de datos SQLite, tu aplicación Node.js recibirá resultados en forma de objetos JavaScript.
    - Puedes manejar estos resultados y aplicar la lógica de negocio necesaria en tu aplicación para procesarlos y tomar decisiones basadas en ellos.
5. **Probar y depurar nuestra aplicación para garantizar su correcto funcionamiento:**
    - Una vez que hayas implementado la lógica de tu aplicación que interactúa con la base de datos SQLite, es importante realizar pruebas exhaustivas para garantizar que todo funcione según lo esperado.
    - Puedes utilizar herramientas de prueba y técnicas de depuración para identificar y corregir cualquier error o comportamiento inesperado en tu aplicación.

Docs_ Generalidades Curiosidades

Este código crea una base de datos llamada **my_database.db** en el directorio actual del proyecto y crea una tabla **usuarios** con una columna **id** y una columna **nombre**.

- **require('sqlite3').verbose()**: Esta línea importa el módulo **sqlite3** y llama a la función **verbose()** para obtener una instancia detallada del módulo que proporciona mensajes de depuración adicionales. Esto nos permite tener un mejor control y visibilidad sobre las operaciones que realizamos con SQLite en Node.js.

- **const path = require('path');**: Se importa el módulo **path** de Node.js, que proporciona utilidades para trabajar con rutas de archivos y directorios. Lo necesitamos para construir la ruta del archivo de la base de datos de forma adecuada.

- **const dbPath = path.resolve(__dirname, 'my_database.db');**: Esta línea construye la ruta completa al archivo de la base de datos. **__dirname** es una variable global de Node.js que contiene la ruta del directorio del script actual. Por lo tanto, **path.resolve(__dirname, 'my_database.db')** construye la ruta absoluta al archivo **my_database.db** en el mismo directorio que el script **database.js**.

- **const db = new sqlite3.Database(dbPath);**: Se crea una nueva instancia de la clase **Database** del módulo **sqlite3** y se pasa la ruta del archivo de la base de datos como argumento. Esto establece una conexión a la base de datos SQLite y nos permite realizar operaciones en ella.

- **db.serialize(() => { ... });**: Esta función permite ejecutar operaciones SQLite de forma síncrona y secuencial dentro de su bloque de código. Esto es útil para asegurarse de que las operaciones se ejecuten en el orden correcto y evita problemas de concurrencia.

- **db.run("CREATE TABLE usuarios (id INTEGER PRIMARY KEY, nombre TEXT)");**: Dentro del bloque **serialize**, ejecutamos una consulta SQL para crear una nueva tabla llamada **usuarios**. Esta tabla tiene dos columnas: **id**, que es un entero y se define como clave primaria, y **nombre**, que es de tipo texto.

- **db.close((err) => { ... });**: Finalmente, cerramos la conexión a la base de datos SQLite una vez que se han completado todas las operaciones. Esto es importante para liberar los recursos y evitar posibles pérdidas de memoria. La función de cierre recibe un callback opcional que se ejecuta una vez que se ha cerrado la conexión. En caso de error al cerrar la conexión, se maneja y se imprime un mensaje de error.

1. **Ejecución del script para crear la base de datos**: Ahora puedes ejecutar este script desde la línea de comandos o desde tu aplicación Node.js para crear la base de datos y la tabla:

node database.js

Esto creará el archivo **my_database.db** en el directorio actual del proyecto y la tabla **usuarios** dentro de esa base de datos.

Docs_ Generalidades Curiosidades

Ejemplo

Aquí tienes un ejemplo completo de cómo utilizar SQLite en un proyecto Node.js. En este ejemplo, crearemos una base de datos SQLite, crearemos una tabla llamada **usuarios**, insertaremos algunos datos en ella, y luego realizaremos una consulta para recuperar esos datos.

Primero, asegúrate de haber instalado el módulo **sqlite3** en tu proyecto utilizando npm:

```
npm install sqlite3
```

Luego, puedes crear un archivo **app.js** y agregar el siguiente código:

```
const sqlite3 = require('sqlite3').verbose();
const path = require('path');

// Ruta del archivo de la base de datos
const dbPath = path.resolve(__dirname, 'my_database.db');

// Crear una instancia de la base de datos
const db = new sqlite3.Database(dbPath);

// Crear la tabla 'usuarios'
db.serialize(() => {
  db.run("CREATE TABLE IF NOT EXISTS usuarios (id INTEGER PRIMARY KEY, nombre TEXT)");

  // Insertar datos de ejemplo
  const insert = db.prepare("INSERT INTO usuarios (nombre) VALUES (?)");
  insert.run("Juan");
  insert.run("María");
  insert.run("Pedro");
  insert.finalize();

  // Consultar todos los usuarios
  db.each("SELECT id, nombre FROM usuarios", (err, row) => {
    if (err) {
      console.error(err.message);
    }
    console.log(row.id + "\t" + row.nombre);
  });
});
```

Docs_ Generalidades Curiosidades

```
// Cerrar la conexión a la base de datos
db.close((err) => {
  if (err) {
    console.error(err.message);
  }
  console.log('Conexión a la base de datos SQLite cerrada');
});
```

Este código realiza las siguientes acciones:

1. Importa el módulo **sqlite3** y el módulo **path**.
2. Define la ruta del archivo de la base de datos.
3. Crea una instancia de la base de datos SQLite.
4. Dentro de **db.serialize()**, crea una tabla llamada **usuarios** si no existe.
5. Inserta algunos datos de ejemplo en la tabla **usuarios**.
6. Realiza una consulta para recuperar todos los usuarios de la tabla y los imprime en la consola.
7. Cierra la conexión a la base de datos.

Para ejecutar este código, simplemente ejecuta **node app.js** en tu terminal. Esto creará la base de datos, insertará datos de ejemplo, recuperará los datos y los imprimirá en la consola.

Docs_ Generalidades Curiosidades

Ejercicio: Gestión de Usuarios con SQLite y Node.js

En este ejercicio, crearás una aplicación Node.js que gestionará una lista de usuarios utilizando una base de datos SQLite. La aplicación permitirá agregar nuevos usuarios, listar todos los usuarios existentes y buscar usuarios por su nombre.

Enunciado:

1. Crea una base de datos SQLite llamada **usuarios.db** con una tabla llamada **usuarios**. Esta tabla debe tener dos columnas: **id** (entero autoincremental) y **nombre** (texto).
2. Crea una aplicación Node.js que permita al usuario realizar las siguientes acciones:
    - Agregar un nuevo usuario.
    - Listar todos los usuarios existentes.
    - Buscar un usuario por su nombre.
3. Implementa las siguientes funciones en tu aplicación:
    - **agregarUsuario(nombre)**: Esta función debería agregar un nuevo usuario a la base de datos con el nombre proporcionado.
    - **listarUsuarios()**: Esta función debería listar todos los usuarios existentes en la base de datos.
    - **buscarUsuarioPorNombre(nombre)**: Esta función debería buscar y mostrar los usuarios cuyo nombre coincida parcial o completamente con el nombre proporcionado.
4. Crea un menú interactivo que permita al usuario seleccionar una de las opciones anteriores y ejecutar la función correspondiente.

Docs_ Generalidades Curiosidades

```
const sqlite3 = require('sqlite3').verbose();
const readline = require('readline').createInterface({
  input: process.stdin,
  output: process.stdout
});

const db = new sqlite3.Database('usuarios.db');

db.serialize(() => {
  db.run("CREATE TABLE IF NOT EXISTS usuarios (id INTEGER PRIMARY KEY, nombre TEXT)");

  // Función para agregar un nuevo usuario
  function agregarUsuario(nombre) {
    const stmt = db.prepare("INSERT INTO usuarios (nombre) VALUES (?)");
    stmt.run(nombre);
    stmt.finalize();
    console.log(`Usuario '${nombre}' agregado.`);
  }

  // Función para listar todos los usuarios
  function listarUsuarios() {
    db.each("SELECT id, nombre FROM usuarios", (err, row) => {
      if (err) {
        console.error(err.message);
      }
      console.log(`${row.id} - ${row.nombre}`);
    });
  }

  // Función para buscar usuarios por nombre
  function buscarUsuarioPorNombre(nombre) {
    db.each("SELECT id, nombre FROM usuarios WHERE nombre LIKE ?", [`%${nombre}%`], (err, row) => {
      if (err) {
        console.error(err.message);
      }
      console.log(`${row.id} - ${row.nombre}`);
    });
  }

  // Menú interactivo
  function mostrarMenu() {
    console.log("\n-- Menú --");
```

Docs_ Generalidades Curiosidades

```
  console.log("1. Agregar usuario");
  console.log("2. Listar usuarios");
  console.log("3. Buscar usuario por nombre");
  console.log("4. Salir");
  readline.question("Seleccione una opción: ", (opcion) => {
   switch (opcion) {
     case '1':
       readline.question("Ingrese el nombre del usuario: ", (nombre) => {
         agregarUsuario(nombre);
         mostrarMenu();
       });
       break;
     case '2':
       console.log("\nListado de usuarios:");
       listarUsuarios();
       mostrarMenu();
       break;
     case '3':
       readline.question("Ingrese el nombre del usuario a buscar: ", (nombre) => {
         console.log("\nUsuarios encontrados:");
         buscarUsuarioPorNombre(nombre);
         mostrarMenu();
       });
       break;
     case '4':
       console.log("Saliendo...");
       readline.close();
       db.close();
       break;
     default:
       console.log("Opción no válida. Por favor, seleccione una opción válida.");
       mostrarMenu();
       break;
   }
  });
 }

 // Iniciar la aplicación mostrando el menú
 mostrarMenu();
});
```

Docs_ Generalidades Curiosidades

Crear una base de datos y una tabla:

```
$ sqlite3 mi_basededatos.db
SQLite version 3.34.1 2021-01-20 14:10:07
Enter ".help" for usage hints.
sqlite> CREATE TABLE usuarios (id INTEGER PRIMARY KEY, nombre TEXT);
```

Insertar datos en la tabla:

```
sqlite> INSERT INTO usuarios (nombre) VALUES ('Juan');

sqlite> INSERT INTO usuarios (nombre) VALUES ('María');
```

Consultar los datos de la tabla:

```
sqlite> SELECT * FROM usuarios;

1|Juan

2|María
```

Actualizar datos en la tabla:

```
sqlite> UPDATE usuarios SET nombre = 'Pedro' WHERE id = 1;
```

Eliminar datos de la tabla:

```
sqlite> DELETE FROM usuarios WHERE id = 2;
```

Consultar el esquema de la tabla:

```
sqlite> .schema usuarios

CREATE TABLE usuarios (id INTEGER PRIMARY KEY, nombre TEXT);
```

Salir de la interfaz de línea de comandos:

```
sqlite> .exit
```