Ejercicio Paso a Paso: AMB de usuarios con Sqlite.

Parte 1

Paso 1: Preparación del proyecto

En este paso inicializamos el proyecto e instalamos la librería `sqlite3` que usaremos para conectarnos a la base de datos

```
mkdir abm_usuarios
cd abm_usuarios
npm init -y
npm i sqlite3
```

Si estamos en linux, creamos el archivo app.js con `touch app.js` . En windows, realizarlo con el explorador de archivos en la misma ruta del proyecto de node (abm_usuarios).

Paso 2: Importar el paquete `sqlite3` y conectarnos a la base de datos

Podemos importar el paquete `sqlite3` como módulo CommonJS (CJS) o como módulo ES (ESM). Veremos ambas alternativas pero nos quedamos con la segunda porque luego vamos a usar otro paquete cuya versión más actual solo admite esta alternativa.
Si querés aprender un poco más sobre la distinción entre CJS y ESM podés ver éste artículo: CommonJS vs ES Modules u otros similares.

Como módulo CommonJS nuestro archivo `app.js` quedaría:

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

En el caso de importarlo cómo módulo ESM tendríamos:

```
import * as sqlite3 from 'sqlite3';
```

Además, este segundo caso necesitamos modificar el archivo `package.json` y agregar está clave:

```
  "type":"module",
```

Ahora vamos a conectarnos a la base de datos. En algunos casos, por ejemplo para testear, podemos preferir utilizar una base de datos 'en memoria' (sin persistencia en disco). En este caso la conexión sería asi:

```
import * as sqlite3 from 'sqlite3';

const db = new sqlite3.Database(':memory:');
```

Si, en cambio, queremos que nuestra base de datos persista en disco, y eso es lo que necesitamos ahora, vamos a especificarlo de esta manera. El archivo de la base de datos en este caso: `usuarios.db` estará en la misma carpeta de nuestro proyecto.

```
import * as sqlite3 from 'sqlite3';

const db = new sqlite3.default.Database("./usuarios.db",
sqlite3.OPEN_READWRITE, (err)=>{
  if (err) return console.error(err.message);
})
```

Si lo ejecutamos por primera vez:

```
node app.js
```

vamos a observar que el archivo usuarios.db ha sido creado.

Paso 3: Crear la tabla `usuarios`

Ahora vamos crear una tabla para almacenar los usuarios. Vamos a crearla con los siguientes campos (podés incluir las variaciones que se te ocurran):

- id
- nombre
- apellido
- usuario
- password
- email

Para crear la tabla vamos a usar la sentencia SQL `CREATE TABLE`. Asignamos la query SQL a una variable y la ejecutamos con el método `run()`

```
import * as sqlite3 from 'sqlite3';

const db = new sqlite3.default.Database("./usuarios.db",sqlite3.OPEN_READWRITE, (err)=>{
  if (err) return console.error(err.message);
})

const sql = "CREATE TABLE usuarios
(id INTEGER PRIMARY KEY, nombre, apellido,
usuario, password, email)";
db.run(sql);
```

Para evitar crear la tabla cada vez que ejectutamos la app, y el consiguiente error porque la tabla ya existe, podemos usar `IF NOT EXISTS`. Asi:

```
import * as sqlite3 from 'sqlite3';

const db = new sqlite3.default.Database("./usuarios.db",sqlite3.OPEN_READWRITE, (err)=>{
  if (err) return console.error(err.message);
})

const sql = "CREATE TABLE IF NOT EXISTS usuarios
(id INTEGER PRIMARY KEY,
nombre, apellido, usuario, password, email)";
db.run(sql);
```

Cuando ejecutemos la aplicación se creará la tabla (si no existe). Podemos verificar esto usando un cliente para sqlite como DBeaver o más sencillo y liviano sqlite3 cli que es probable que ya tengas instalado en tu sistema operativo.

Paso 4: Insertar filas

En este punto ya podemos inspeccionar el contenido de la tabla ... pero la tabla está vacía. Entonces vamos a comenzar creando alguna o algunas filas en la tabla usando la sentencia SQL: `insert`

```
import * as sqlite3 from 'sqlite3';

const db = new sqlite3.default.Database("./usuarios.db",sqlite3.OPEN_READWRITE, (err)=>{
  if (err) return console.error(err.message);
})

const sql = "CREATE TABLE IF NOT EXISTS usuarios
(id INTEGER PRIMARY KEY,
nombre, apellido, usuario, password, email)";
db.run(sql);

const sql_insert = "INSERT INTO usuarios (nombre, apellido,
usuario, password, email) VALUES (?,?,?,?,?)";
const params = ['Juan', 'Perez', 'jperez', 'secreto', 'jperez@example.com'];
db.run (sql_insert, params, (err) => {
  if (err) return console.error(err.message)
})
```

Cada vez que ejecutemos nuestra aplicación se insertará una fila con los mismos datos, de modo que podemos variar algunos de ellos de modo de tener filas distintas. Más adelante vamos a agregar un poco de interactividad a nuestra pequeña aplicación, pero por ahora podemos hacerlo de esta forma rudimentaria.

Paso 5: Listar el contenido de la tabla

Ahora con algunos usuarios en nuestra tabla de usuarios vamos a listar todas las filas de la tabla:
Podés comentar o eliminar las líneas de código que insertan usuarios de modo de no terminar con tantas filas iguales.

```
import * as sqlite3 from 'sqlite3';

const db = new sqlite3.default.Database("./usuarios.db",sqlite3.OPEN_READWRITE, (err)=>{
  if (err) return console.error(err.message);
})

const sql = "CREATE TABLE IF NOT EXISTS usuarios (id INTEGER PRIMARY KEY,
nombre, apellido, usuario, password, email)";
db.run(sql);

const sql_select = "SELECT * FROM usuarios"
db.all(sql_select, [], (err,rows)=>{
  if(err) return console.error(err.message);
  rows.forEach((row)=>{
    console.log(row);
  });
});
```

Paso 6: Modificar una fila

Ahora vamos a implementar la operación de modificación de una fila de la tabla usuarios. En este caso vamos a modificar el valor de `usuario` en la fila cuyo `id` es 2.

```
import * as sqlite3 from 'sqlite3';

const db = new sqlite3.default.Database("./usuarios.db",sqlite3.OPEN_READWRITE, (err)=>{
  if (err) return console.error(err.message);
})

const sql = "CREATE TABLE IF NOT EXISTS
usuarios (id INTEGER PRIMARY KEY,
nombre, apellido, usuario, password, email)";
db.run(sql);

const sql_update = "UPDATE usuarios SET usuario = ? WHERE id = ?";
const params = ['jperez_modificado', 2];
db.run (sql_update, params, (err) => {
  if (err) return console.error(err.message);
})

const sql_select = "SELECT * FROM usuarios"
db.all(sql_select, [], (err,rows)=>{
  if(err) return console.error(err.message);
  rows.forEach((row)=>{
    console.log(row);
  });
});
```

Paso 7: Eliminar una fila

Por último para completar las cuatro operaciones, vamos a ver como remover o eliminar una fila. Haremos esto con la fila que acabamos de modificar, la que tiene `id` 2.

```
import * as sqlite3 from 'sqlite3';

const db = new sqlite3.default.Database("./usuarios.db",
sqlite3.OPEN_READWRITE, (err)=>{
  if (err) return console.error(err.message);
})

const sql = "CREATE TABLE
IF NOT EXISTS usuarios (id INTEGER PRIMARY KEY,
nombre, apellido, usuario, password, email)";
db.run(sql);

const sql_delete = "DELETE FROM usuarios WHERE id = ?";
const params = [2];
db.run (sql_delete, params, (err) => {
  if (err) return console.error(err.message);
})

const sql_select = "SELECT * FROM usuarios"
db.all(sql_select, [], (err,rows)=>{
  if(err) return console.error(err.message);
  rows.forEach((row)=>{
    console.log(row);
  });
});
```