Conceptos avanzados (práctica)

Tiempo estimado: 15min

El objeto de esta práctica es afianzar, reforzar y consolidar los conocimientos teóricos presentados en la lección.

Al finalizarla, el estudiante:

- Habrá creado y trabajado con una base de datos en memoria.
- Habrá usado la caché de conexiones.
- Habrá serializado comandos SQL.

Objetivos

El objetivo de la práctica es mostrar cómo trabajar con aspectos avanzados de SQLite. Para ello, crearemos una base de datos en memoria y usaremos la base de datos en disco de la práctica anterior con conexiones cacheadas.

Base de datos en memoria

Comencemos trabajando con una base de datos en memoria:

1. Abrir una consola.
2. Ir al directorio de la práctica que creamos en la prática anterior.
3. Abrir node en modo interactivo.
4. Importar el *driver*:

```
> const sqlite = require("sqlite3")
undefined
>
```

5. Crear una base de datos en memoria:

```
> db = new sqlite.Database(":memory:", function(err) { if (err)
console.error(err); })
Database { open: false, filename: ':memory:', mode: 65542 }
> db
Database { open: true, filename: ':memory:', mode: 65542 }
>
```

6. Comprobar que no se ha creado ningún archivo en disco:

```
> fs.readdirSync(".")
[ 'mibd.db', 'node_modules' ]
>
```

7. Crear una tabla en la base de datos e insertar una fila:

```
> db.run("CREATE TABLE Tabla(x, y)", function(err) {
... if (err) return console.error(err);
... db.run("INSERT INTO Tabla(x, y) VALUES(1, 2)", function(err) {
..... if (err) console.error(err);
..... });
... });
Database { open: true, filename: ':memory:', mode: 65542 }
>
```

8. Consultar la tabla de prueba:

```
> db.all("SELECT * FROM Tabla", function(err, rows) {
... if (err) console.error(err);
... else console.dir(rows);
... });
Database { open: true, filename: ':memory:', mode: 65542 }
```



```
> [ { x: 1, y: 2 } ]
```

9. Abrir otra conexión a la base de datos en memoria:

```
> db2 = new sqlite.Database(":memory:", function(err) { if (err)
console.error(err); })
Database { open: false, filename: ':memory:', mode: 65542 }
> db2
Database { open: true, filename: ':memory:', mode: 65542 }
>
```

10. Consultar la tabla de prueba:

```
> db2.all("SELECT * FROM Tabla", function(err, rows) { if (err)
console.error(err); else console.dir(rows); })
Database { open: true, filename: ':memory:', mode: 65542 }
> { Error: SQLITE_ERROR: no such table: Tabla errno: 1, code: 'SQLITE_ERROR' }
```

En efecto, la tabla no existe, porque las bases de datos en memoria son específicas y particulares de cada conexión. No se puede abrir dos conexiones a la misma base de datos en memoria.

Conexiones cacheadas

Recordemos que la caché de conexiones es un almacén interno en el que el *driver* registra las conexiones según su base de datos en disco. De tal manera que cada vez que se accede a una base de datos, se pueda reutilizar conexiones registradas, reduciendo los procesos de apertura. Para usarla, hay que usar la propiedad cached del *driver*.

1. Ir a la consola.
2. Abrir una conexión, no cacheada, a la base de datos mibd.db:

```
> db = new sqlite.Database("mibd.db", function(err) { if (err) console.error(err);
})
Database { open: false, filename: 'mibd.db', mode: 65542 }
>
```

3. Consultar los datos de la tabla de prueba:

```
> db.all("SELECT * FROM Tabla", function(err, rows) { if (err) console.error(err);
else console.dir(rows); })
Database { open: true, filename: 'mibd.db', mode: 65542 }
> [ { x: 1, y: 2, z: 3 },
  { x: 3, y: 2, z: 1 },
  { x: 123, y: 456, z: 789 },
  { x: 321, y: 654, z: 987 },
  { x: 135, y: 246, z: 789 },
  { x: 531, y: 642, z: 987 } ]
```

4. Abrir una nueva conexión y consultar los datos:

```
> db2 = new sqlite.Database("mibd.db", function(err) { if (err)
console.error(err); })
Database { open: false, filename: 'mibd.db', mode: 65542 }
> db2.all("SELECT * FROM Tabla", function(err, rows) { if (err)
console.error(err); else console.dir(rows); })
Database { open: true, filename: 'mibd.db', mode: 65542 }
> [ { x: 1, y: 2, z: 3 },
  { x: 3, y: 2, z: 1 },
  { x: 123, y: 456, z: 789 },
  { x: 321, y: 654, z: 987 },
  { x: 321, y: 654, z: 987654 },
  { x: 135, y: 246, z: 789 },
  { x: 531, y: 642, z: 987 } ]
```

5. Comparar las instancias:

```
> db
Database { open: true, filename: 'mibd.db', mode: 65542 }
> db2
Database { open: true, filename: 'mibd.db', mode: 65542 }
> db === db2
false
>
```


Son objetos distintos.

6. Abrir una conexión cacheada:

```
> db = new sqlite.cached.Database("midb.db", function(err) { if (err)
console.error(err); })
Database {
  open: false,
  filename: 'midb.db',
  mode: 65542 }
> db
Database {
  open: true,
  filename: '/home/me/tests/sqlite/driver/midb.db',
  mode: 65542 }
>
```

7. Abrir una nueva conexión cacheada:

```
> db2 = new sqlite.cached.Database("midb.db", function(err) { if (err)
console.error(err); })
Database {
  open: true,
  filename: 'midb.db',
  mode: 65542 }
>
```

8. Comprobar que las instancias de las variables db y db2 son la misma:

```
> db === db2
true
>
```

9. Consultar la propiedad cached del *driver*:

```
> sqlite.cached
```

10. Cerrar la conexión cacheada:

```
> db.close(function(err) { if (err) console.error(err); })
Database {
  open: true,
  filename: 'midb.db',
  mode: 65542 }
> db
Database {
  open: false,
  filename: 'midb.db',
  mode: 65542 }
>
```

11. Abrir una nueva conexión cacheada:

```
> db3 = new sqlite.cached.Database("midb.db", function(err) { if (err)
console.error(err); })
Database {
  open: false,
  filename: '/home/me/tests/sqlite/driver/midb.db',
  mode: 65542,
  _events: { open: { [Function: g] listener: [Function: cb] } },
  _eventsCount: 1 }
> db === db3
true
> db2 === db3
true
>
```

Cuidado con cerrar conexiones cacheadas. Las nuevas conexiones reutilizarán la conexión cerrada y no podrán ejecutar comandos SQL contra la base de datos.

Serialización de comandos

Ahora, vamos a serializar comandos SQL mediante una función:

1. Ir a la consola.

2. Abrir una conexión:

```
> db = new sqlite.Database("mibd.db", function(err) { if (err) console.error(err);
})
Database { open: false, filename: 'mibd.db', mode: 65542 }
> db
Database { open: true, filename: 'mibd.db', mode: 65542 }
>
```

3. Insertar nuevas filas de manera serializada mediante una función:

```
> db.serialize(function(err) {
... db.run("INSERT INTO Tabla(x, y, z) VALUES(111, 222, 333);");
... db.run("INSERT INTO Tabla(x, y, z) VALUES(444, 555, 666);");
... })
Database { open: true, filename: 'mibd.db', mode: 65542 }
>
```

4. Comprobar el contenido de la tabla de prueba:

```
> db.all("SELECT * FROM Tabla", function(err, rows) { if (err) console.error(err);
else console.dir(rows); })
Database { open: true, filename: 'mibd.db', mode: 65542 }
> [ { x: 1, y: 2, z: 3 },
  { x: 3, y: 2, z: 1 },
  { x: 123, y: 456, z: 789 },
  { x: 321, y: 654, z: 987 },
  { x: 135, y: 246, z: 789 },
  { x: 531, y: 642, z: 987 },
  { x: 111, y: 222, z: 333 },
  { x: 444, y: 555, z: 666 } ]
```

5. Cerrar la conexión:

```
> db.close(function(err) { if (err) console.error(err); })
Database { open: true, filename: 'mibd.db', mode: 65542 }
> db
Database { open: false, filename: 'mibd.db', mode: 65542 }
>
```

