codecademy

Learn Node-SQLite

sqlite3 Module

Node.js and SQLite are separate entities, but both can be used together to create powerful applications. First, we need to link them together by requiring the **sqlite3** module. Once we have required this module in our JavaScript, it will give us access to methods to perform various actions with the database.

```
//requiring the sqlite3 module

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

Creating/Connecting To A Database

In SQLite, a database corresponds to a single file. The `sqlite3.Database()` method can be used to create or connect to a single file.

```
// creating a database file
const db = new
sqlite3.Database('./db.sqlite');
```

db.get() Method

Sometimes we only need one row from a database. The method `db.get()` allows us to fetch a single row from a database matching a query. If multiple rows match the query only one row matching the query will be returned.

```
// This will return the first row
matching the query.
db.get("SELECT * FROM drinks WHERE type =
'soda'")
```

db.all() Method

A query is a statement that speaks to a database and requests specific information from it. The `db.all()` method allows us to execute a query that returns all rows with specific data. For example, we can use `db.all()` to return all rows from a table that has the pet as a cat.

```
// Selects a table named Animal and
returns only the rows the has the pet as
a cat
db.all("SELECT * FROM Animal WHERE pet =
'cat'")
```

db.each() Method

Sometimes we want to perform an action every time a row is returned. Using the `db.each()` method we can do exactly that. `db.each()` takes a query and a callback function that it performs on each row returned from the query. `db.each()` can also take a second callback function, which will be called when all of the queries are completed and processed.

```
db.each("SELECT * FROM Sports WHERE type
= 'baseball'",
  (error, row) => {
  // This will be printed everytime a row
is returned
  console.log(`${row.name} is a good
baseball team`);
};
```

db.run() Method

Sometimes we want to do more than just get a result from a database. The `db.run()` method holds SQL commands that do not return rows; such as commands that will allow us to create, update tables, or insert new rows.

```
//creating a table
db.run("CREATE TABLE user (id INT, dt
TEXT)");

//updating existing table
db.run('INSERT INTO Key (type, color)
VALUES ($type, $color)
```

db.serialize() Method

Any requests we send to our database get processed as quickly as possible, which can lead to multiple requests processing at the same time. Usually, this creates efficiency but in some cases like when we want to create a table and insert rows, it can cause errors. This is because our request might try to insert a row into a table that's not even created yet. This problem can be solved by the `db.serialize()` method which takes multiple requests and executes them one by one.

```
//each request inside this method will be
executed one by one
db.serialize(() => {
  db.run("DROP TABLE Stocks");
  db.run("CREATE TABLE Stocks");
  db.run("INSERT INTO Stocks (AMD, MSFT,
TSLA);
});
```

Handling Errors

sqlite3 uses Node.js error-first callback style.The first argument in the methods `db.run()`, `db.each()`, `db.get()`, and `db.all()` will always be an Error object. This object will return an error if it exists and return `null` if no errors are found.

```
// an if statement can be used to log the
error
if (err) {
        console.log(err);
}
```