NodeJS and SQLite

CSC 342 - Web Technologies

Node.js and SQLite

- The `sqlite3` library defines an interface for accessing SQLite databases from a Node.js application.
- The full documentation is here: SQLite3 API
- The library needs to be installed to use it in a project:

```
mkdir myapp
cd myapp
npm init
npm install sqlite3 --save
```

- The import is:

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

Connecting to a SQLite Database

- The API to connect to a SQLite database is:

  ```
  new sqlite3.Database(filename, [mode],
  [callback])
  ```

  - `filename`: path to the SQLite database file or `:memory:` for in in-memory database.
  - `mode` (optional): permissions – one or more of the following
    - `sqlite3.OPEN_READONLY`
    - `sqlite3.OPEN_READWRITE`
    - `sqlite3.OPEN_CREATE`
    - Default value: `sqlite3.OPEN_READWRITE | sqlite3.OPEN_CREATE`
  - `callback` (optional): callback to call when the database is opened successfully or when an error ocurrs.

Example: Connect to a SQLite Database

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

let db = new sqlite3.Database('./my.db', (err) => {
    if (err) {
        console.log('ERROR: ' + err);
        exit(1);
    }
});

// run queries or modify database contents

db.close();
```

Execute a SQL Statement

- The API to run a query is:

  `db.run(sql, [param, ...], [callback])`

  - `sql`: the SQL query to run
  - `param` (optional): arguments for placeholders
  - `callback` (optional): signature (err, result) called when an error occurs; if execution was successful, the `this` object contains the properties `lastID` and `changes`

Example: Execute a SQL Statement

```
let sql =
'INSERT INTO user (name, email, password) VALUES (?,?,?)';

let params = ['Bob', 'bob@bob.com', 'swordfish'];

db.run(sql, params, (err, result) => {
    if (err) {
        // do error stuff
    }
    // possibly do something with the result
});
```

Run a SQL Query

- The API to run a function on every result from a query is:

  `db.all(sql, [param, ...], [callback]`

  - `sql`: the SQL query to run
  - `param` (optional): arguments for placeholders
  - `callback` (optional): signature (err, rows) called when an error occurs; if the query was successful, the rows is an array

Example: Run a SQL Query

```
let sql = 'SELECT * FROM user WHERE name = ?';
let params = ['Bob'];

db.all(sql, params, (err, rows) => {
    if (err) {
        // do error stuff
    }
    rows.forEach(row => console.log(row));
});
```