Лабораторная работа №5

**Тема**: «Работа с базами данных в Node.js»

**Цель**: изучить основы работы по написанию скриптов для работы с базами данных в Node.js. Получить навыки работы с orm в Node.js. Научиться создавать и тестировать простые приложения.

**Теоретический материал**:

Для взаимодействия с базой данных, в примере будет показана база SQLite вам будет необходимо загрузить и установить соответствующий модуль для sqlite - sqlite3. Вы можете использовать npm для этого, используя следующую команду



Рисунок 1 – консольная команда

После установки модуля sqlite3 можно подключиться к базе данных SQLite из Node.js приложение. Что бы подключиться к базе данных SQLite, вам необходимо:

Сначала импортируйте модуль sqlite3

Во-вторых, вызовите функцию Database() модуля sqlite3 и передайте информацию о базе данных, такую как файл базы данных, режим открытия и функция обратного вызова.

let db = new sqlite3.Database('./../database.db', (err) => {

if (err) {

console.error(err.message);

}

console.log('Успешное подключение к базе данных.');

});

Пример кода 1 - Подключения к базе данных

Есть три режима открытия:

1. sqlite3.OPEN\_READONLY: открыть базу данных только для чтения.
2. sqlite3.OPEN\_READWRITE: открыть базу данных для чтения и записи.
3. sqlite3.OPEN\_CREATE: открыть базу данных, если база данных не существует, создать новую базу данных.

В sqlite3.Database() качестве второго аргумента принимает один или несколько режимов. По умолчанию используется OPEN\_READWRITE | OPEN\_CREATE режим. Это означает, что, если база данных не существует, новая база данных будет создана и готова для чтения и записи.

Чтобы открыть database образец базы данных для чтения и записи, вы можете сделать это следующим образом:

let db = new sqlite3.Database('./../database.db', sqlite3.OPEN\_READWRITE, (err) => {

if (err) {

console.error(err.message);

}

console.log('Успешное подключение к базе данных.');

});

Пример кода 2 - Подключения к базе данных c режимом

В следующем примере показан полный код для открытия database базы данных, запроса данных из playlists таблицы и закрытия соединения с базой данных.

const sqlite3 = require('sqlite3');

// open the database

let db = new sqlite3.Database('./../database.db', sqlite3.OPEN\_READWRITE, (err) => {

if (err) {

console.error(err.message);

}

console.log('Успешное подключение к базе данных.');

});

db.serialize(() => {

db.each(`SELECT PlaylistId as id,

Name as name

FROM playlists`, (err, row) => {

if (err) {

console.error(err.message);

}

console.log(row.id + "\t" + row.name);

});

});

db.close((err) => {

if (err) {

console.error(err.message);

}

console.log('Закрытие базы данных');

});

Пример кода 3 – Выборка в базе данных

Модуль sqlite3предоставляет вам некоторые методы для запроса данных, такие как all(), each()и get(). All() – выбирает все строки таблицы, each и get – параметризованные.

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

// open the database

let db = new sqlite3.Database('./db/chinook.db');

let sql = `SELECT DISTINCT Name name FROM playlists

ORDER BY name`;

db.all(sql, [], (err, rows) => {

if (err) {

throw err;

}

rows.forEach((row) => {

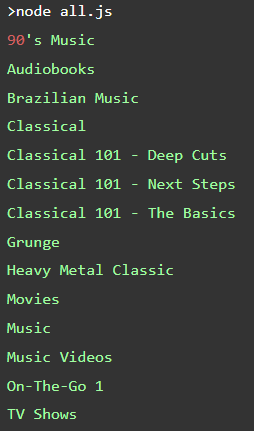
console.log(row.name);

});

});

// close the database connection

db.close();



Результат работы метода all

Метод serialize() переводит режим выполнения в сериализованный режим. Это означает, что одновременно может выполняться только один оператор. Другие операторы будут ждать в очереди, пока не будут выполнены все предыдущие операторы. После serialize() возврата метода режим выполнения снова устанавливается в исходный режим.

Предположим, вы хотите последовательно выполнить следующие три оператора:

1. Создайте новую таблицу.
2. Внесите данные в таблицу.
3. Запросить данные из таблицы.

Для этого вы помещаете эти операторы в serialize() метод следующим образом:

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

// open the database connection

let db = new sqlite3.Database(':memory:', (err) => {

if (err) {

console.error(err.message);

}

});

db.serialize(() => {

// Queries scheduled here will be serialized.

db.run('CREATE TABLE greetings(message text)')

.run(`INSERT INTO greetings(message)

VALUES('Hi'),

('Hello'),

('Welcome')`)

.each(`SELECT message FROM greetings`, (err, row) => {

if (err){

throw err;

}

console.log(row.message);

});

});

// close the database connection

db.close((err) => {

if (err) {

return console.error(err.message);

}

});

Пример кода 4 – Использование метода serialize()

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

// open the database connection

let db = new sqlite3.Database('../db/sample.db');

let languages = ['C++', 'Python', 'Java', 'C#', 'Go'];

// construct the insert statement with multiple placeholders

// based on the number of rows

let placeholders = languages.map((language) => '(?)').join(',');

let sql = 'INSERT INTO langs(name) VALUES ' + placeholders;

// output the INSERT statement

console.log(sql);

db.run(sql, languages, function(err) {

if (err) {

return console.error(err.message);

}

console.log(`Rows inserted ${this.changes}`);

});

// close the database connection

db.close();

Пример кода 5 – Добавление в множество параметров в бузу данных

// Display a single user by ID

app.get('/users/:id', (request, response) => {

const id = request.params.id;

db.get(sql, [id], (err, row) => {

if (err) {

return console.error(err.message);

}

response.send( row ? row: {“message”: error});

});

});

Пример кода 6 – Отправка пользователя по указанному id

Использование ORM на примере seqlize:

const { Sequelize } = require('@sequelize/core');

// Option 1: Passing a connection URI

const sequelize = new Sequelize('sqlite::memory:') // Example for sqlite

const sequelize = new Sequelize('postgres://user:pass@example.com:5432/dbname') // Example for postgres

// Option 2: Passing parameters separately (sqlite)

const sequelize = new Sequelize({

dialect: 'sqlite',

storage: 'path/to/database.sqlite'

});

// Option 3: Passing parameters separately (other dialects)

const sequelize = new Sequelize('database', 'username', 'password', {

host: 'localhost',

dialect: /\* one of 'mysql' | 'mariadb' | 'postgres' | 'mssql' \*/

});

Пример кода 7 – подключение к seqlize

const sequelize = new Sequelize('sqlite::memory:');

class User extends Model {}

User.init({

// Model attributes are defined here

firstName: {

type: DataTypes.STRING,

allowNull: false

},

lastName: {

type: DataTypes.STRING

// allowNull defaults to true

}

}, {

// Other model options go here

sequelize, // We need to pass the connection instance

modelName: 'User' // We need to choose the model name

});

// the defined model is the class itself

console.log(User === sequelize.models.User); // true

Пример кода 8 – Определение моделей

const project = await Project.findOne({ where: { title: 'My Title' } });

if (project === null) {

console.log('Not found!');

} else {

console.log(project instanceof Project); // true

console.log(project.title); // 'My Title'

}

Пример кода 9 – Поиск по одной записи

**Практическая часть:**

**Задание 1** – Реализовать Rest Api (добавлении, удаление, обновление, получение) с использование express и базы данных. Предметную область взять по усмотрению студента. Использовать sql запросы к обращению к базе данных.

**Задание 2** – Повторить задание 1, но с использование ORM.

**Задание 3** – Протестировать ранее созданное Api используя библиотеки для тестов – Jest, Mocha, supertest.

**Задание 4** – Реализовать паттерн MVC, создать три папки в которых будут хранится классы моделей, контроллеров и представлений. Для представлений можно использовать handlebars. Классы сервисов и маршрутов находятся в отдельных файлах.

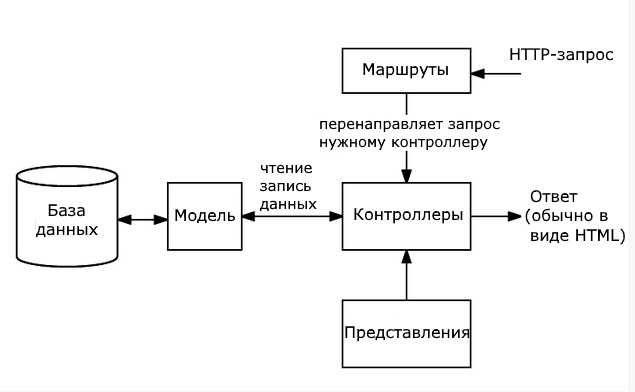


Рисунок 3 – Схема паттерна

**Ссылки для детального просмотра**:

Для работы с sqlite - <https://www.sqlitetutorial.net/sqlite-nodejs/>

Для работы с mysql - <https://code.tutsplus.com/ru/tutorials/code-your-first-api-with-nodejs-and-express-connect-a-database--cms-31699>

ORM Seqlize - <https://sequelize.org/v6/>

Тестирование Api с Jest - <https://www.albertgao.xyz/2017/05/24/how-to-test-expressjs-with-jest-and-supertest/>