



Clase 16. Programación Backend

SQL y Node.js



OBJETIVOS DE LA CLASE

- Integrar dependencia Knex para habilitar a Node.js como cliente de base de datos.
- Interactuar con MySQL / MariaDB a través de NodeJS y Knex
- Configurar Knex para trabajar con SQLite3

CRONOGRAMA DEL CURSO

Clase 15



SQL

Clase 16



SQL y Node.js

Clase 17



Introducción a MongoDB

Node.js como cliente de MySQL / MariaDB



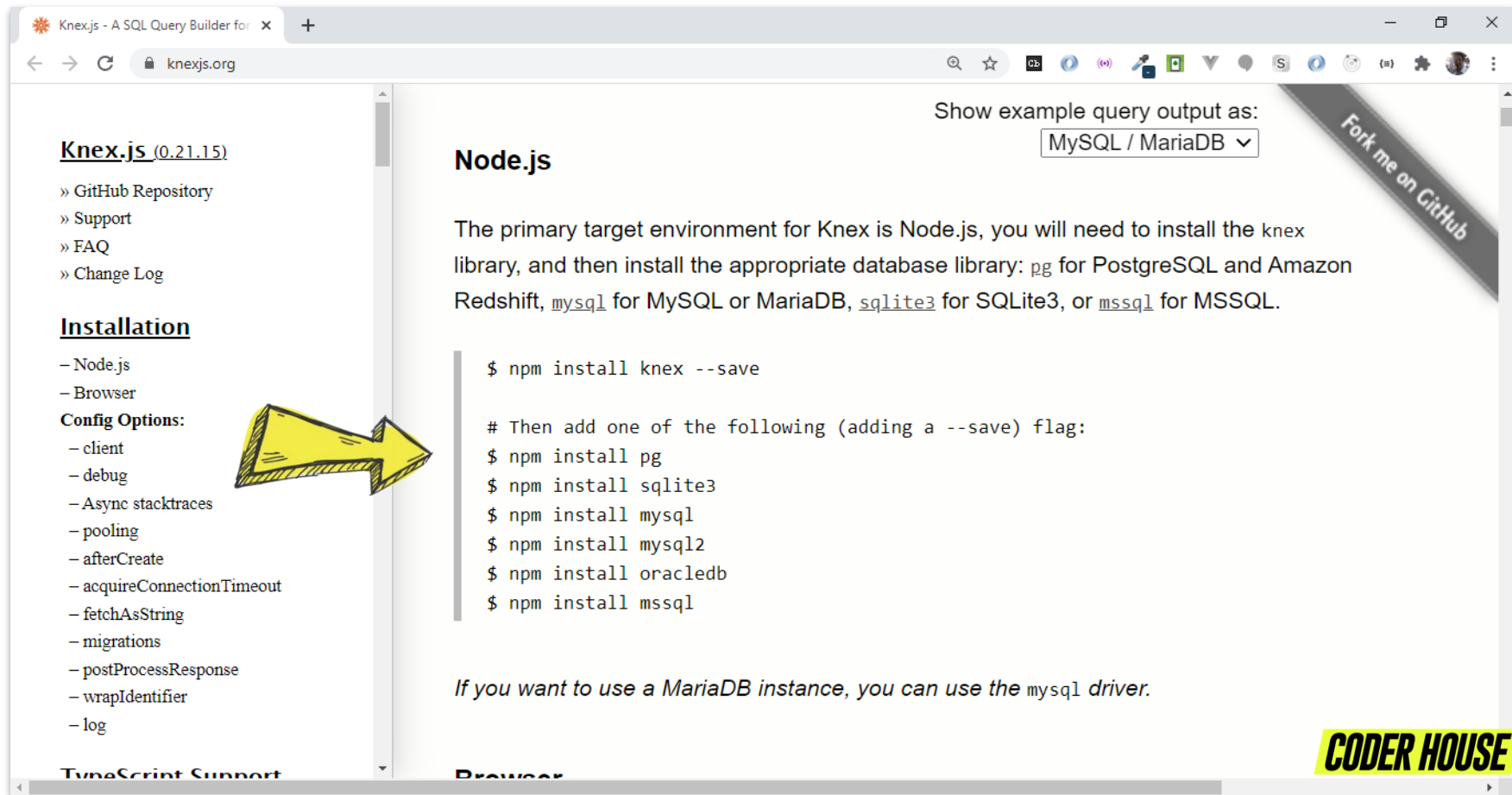
KNEX.JS

¿Qué es Knex.js?



- Knex.js es un **generador de consultas SQL** con "baterías incluidas" para *Postgres, MSSQL, MySQL, MariaDB, SQLite3, Oracle y Amazon Redshift*, diseñado para ser flexible, portátil y fácil de usar.
- Cuenta con una **interfaz basada en callbacks y en promesas**.
- Knex se puede utilizar como un generador de consultas SQL en Node.JS.
- Se puede instalar desde npm con el comando ***npm i knex***
- Además debemos instalar las dependencias de las base de datos con la cual vamos a trabajar: ***npm i -> pg*** para *PostgreSQL* y *Amazon Redshift*, ***mysql*** para *MySQL* y *MariaDB*, ***sqlite3*** para *SQLite3* ó ***mssql*** para *MSSQL*

KNEX.JS Instalación en Node.js



Knex.js (0.21.15)

- » GitHub Repository
- » Support
- » FAQ
- » Change Log

Installation

- Node.js
- Browser

Config Options:

- client
- debug
- Async stacktraces
- pooling
- afterCreate
- acquireConnectionTimeout
- fetchAsString
- migrations
- postProcessResponse
- wrapIdentifier
- log

Node.js

The primary target environment for Knex is Node.js, you will need to install the `knex` library, and then install the appropriate database library: `pg` for PostgreSQL and Amazon Redshift, `mysql` for MySQL or MariaDB, `sqlite3` for SQLite3, or `mssql` for MSSQL.

Show example query output as: MySQL / MariaDB ▾

```
$ npm install knex --save
```

Then add one of the following (adding a --save) flag:

```
$ npm install pg
$ npm install sqlite3
$ npm install mysql
$ npm install mysql2
$ npm install oracledb
$ npm install mssql
```

If you want to use a MariaDB instance, you can use the `mysql` driver.

Browser

CODER HOUSE

KNEX.JS Cheatsheet [***https://devhints.io/knex***](https://devhints.io/knex)

Knex cheatsheet

devhints.io/knex

DEVHINTS.IO

←

→

Edit

Knex cheatsheet

Knex is an SQL query builder for Node.js. This guide targets v0.13.0.

Connect

```
require('knex')({
  client: 'pg',
  connection: 'postgres://user:pass@localhost:5...'
})
```

See: [Connect](#)

Create table

```
knex.schema.createTable('user', (table) => {
  table.increments('id')
  table.string('name')
  table.integer('age')
})
.then(() => ...)
```

See: [Schema](#)

Select

```
knex('users')
  .where({ email: 'hi@example.com' })
  .then(rows => ...)
```

See: [Select](#)

Update

```
knex('users')
  .where({ id: 135 })
  .update({ email: 'hi@example.com' })
```

Migrations

```
knex init
knex migrate:make migration_name
knex migrate:make migration_name -x ts # Genera...
```

Insert

```
knex('users')
  .insert({ email: 'hi@example.com' })
```

See: [Insert](#)

Jira Software

From to do to done with Jira Software
ads via Carbon

CODER HOUSE



Knex: Resumen de comandos

Connect

```
require('knex')({
  client: 'pg',
  connection: 'postgres://user:pass@localhost:5432/db'
})
```

See: [Connect](#)

Update

```
knex('users')
  .where({ id: 135 })
  .update({ email: 'hi@example.com' })
```

See: [Update](#)

Create table

```
knex.schema.createTable('user', (table) => {
  table.increments('id')
  table.string('name')
  table.integer('age')
})
.then(() => ...)
```

See: [Schema](#)

Migrations

```
knex init
knex migrate:make migration_name
knex migrate:make migration_name -x ts # Generates a TypeScript migration
knex migrate:latest
knex migrate:rollback
```

See: [Migrations](#)

Select

```
knex('users')
  .where({ email: 'hi@example.com' })
  .then(rows => ...)
```

See: [Select](#)

Insert

```
knex('users')
  .insert({ email: 'hi@example.com' })
```

See: [Insert](#)

Seeds

```
knex seed:make seed_name
knex seed:make seed_name -x ts # Generates a TypeScript seed file
knex seed:run # Runs all seed files
knex seed:run --specific=seed-filename.js # Run a specific seed
```

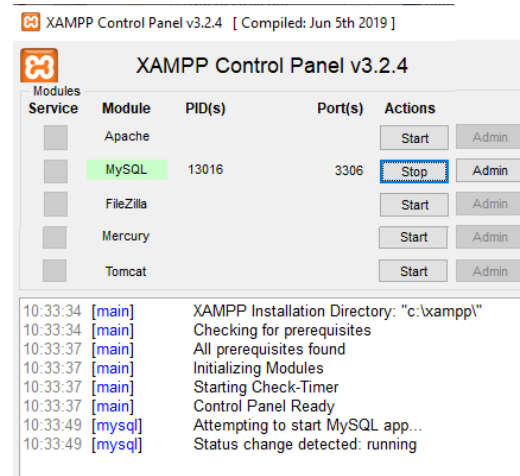
See: [Seeds](#)

Proyecto Cliente MySQL/MariaDB Node.js con Knex.js

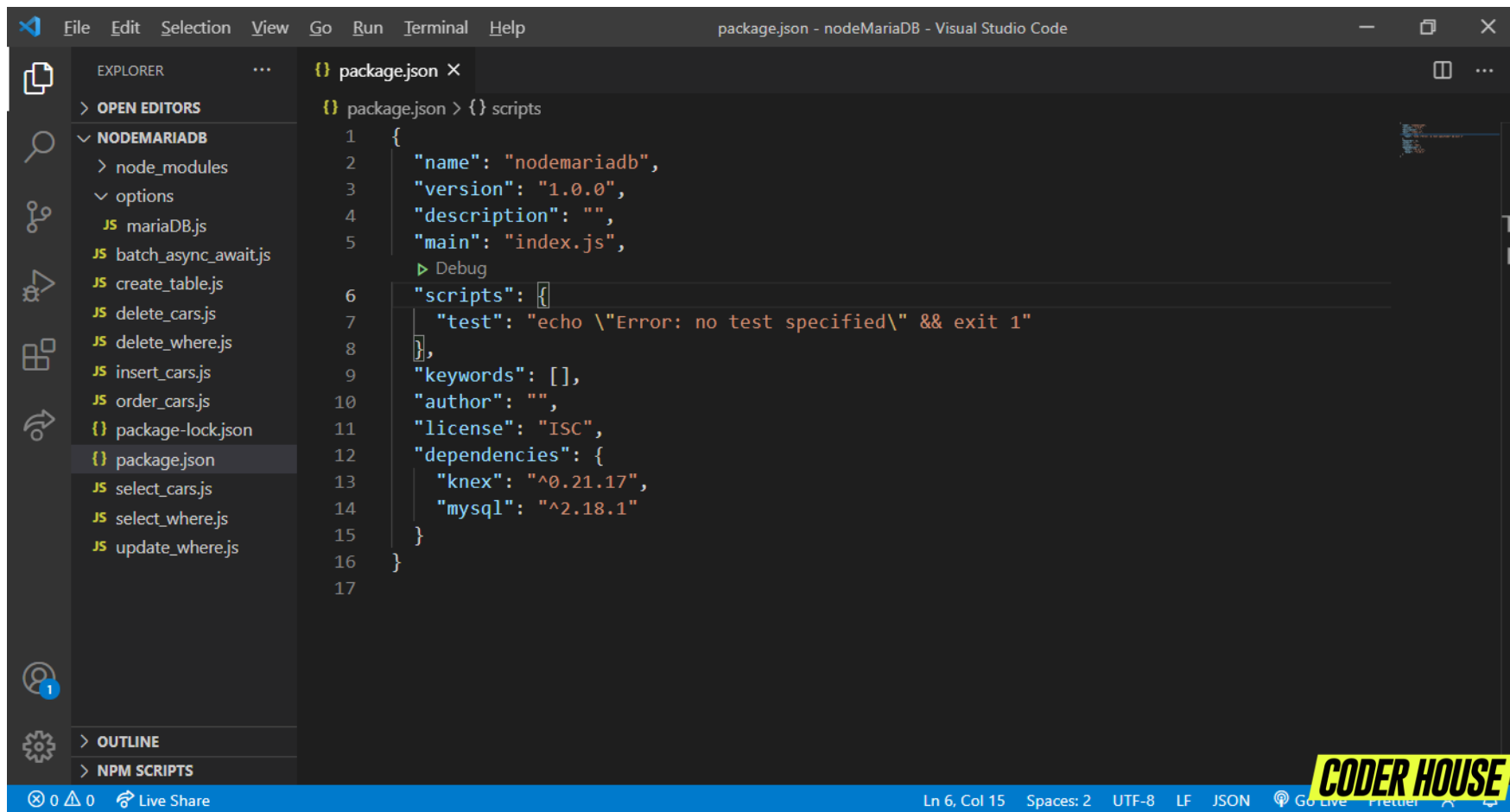


Inicialización del proyecto e instalación de dependencias

1. Creamos un proyecto Node.js con **npm init -y**
2. Instalamos la dependencias *Knex* y *mysql* con **npm i knex mysql** (mysql es el plugin necesario para trabajar con *MariaDB*)
3. Levantamos el motor de base de datos *MariaDB* con *XAMPP*.
4. Creamos los archivos necesarios para probar los comandos SQL necesarios en acciones CRUD.



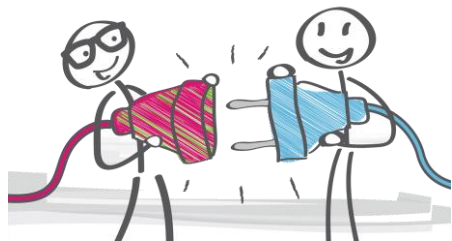
Proyecto Knex MariaDB: package.json



The image shows a Visual Studio Code editor window with the file 'package.json' open. The Explorer sidebar on the left shows the project structure, including 'node_modules', 'options', and several JavaScript files. The main editor area displays the content of 'package.json' with line numbers 1 through 17. The file is a JSON object with the following properties: 'name' (nodemariadb), 'version' (1.0.0), 'description' (empty), 'main' (index.js), 'scripts' (a test script), 'keywords' (empty array), 'author' (empty), 'license' (ISC), and 'dependencies' (knex and mysql).

```
1 {  
2   "name": "nodemariadb",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "index.js",  
6   "scripts": {  
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"  
8   },  
9   "keywords": [],  
10  "author": "",  
11  "license": "ISC",  
12  "dependencies": {  
13    "knex": "^0.21.17",  
14    "mysql": "^2.18.1"  
15  }  
16 }  
17
```

The status bar at the bottom indicates the current position is Line 6, Column 15, with 2 spaces, in UTF-8 encoding, LF line endings, and JSON format. A 'Live Share' icon is also visible on the left.



Knex: Conexión a MariaDB y a las distintas bases de datos

Libraries

pg	PostgreSQL
mysql	MySQL or MariaDB
sqlite3	Sqlite3
mssql	MSSQL

Install any of these packages along with knex.

See: [Node.js installation](#)

Connect via host

```
var knex = require('knex')({
  client: 'mysql',
  connection: {
    host: '127.0.0.1',
    user: 'your_database_user',
    password: 'your_database_password',
    database: 'myapp_test'
  },
  pool: { min: 0, max: 7 }
})
```

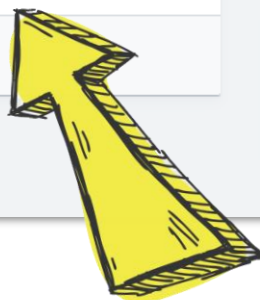
See: [Initializing the library](#)

Connect via URL

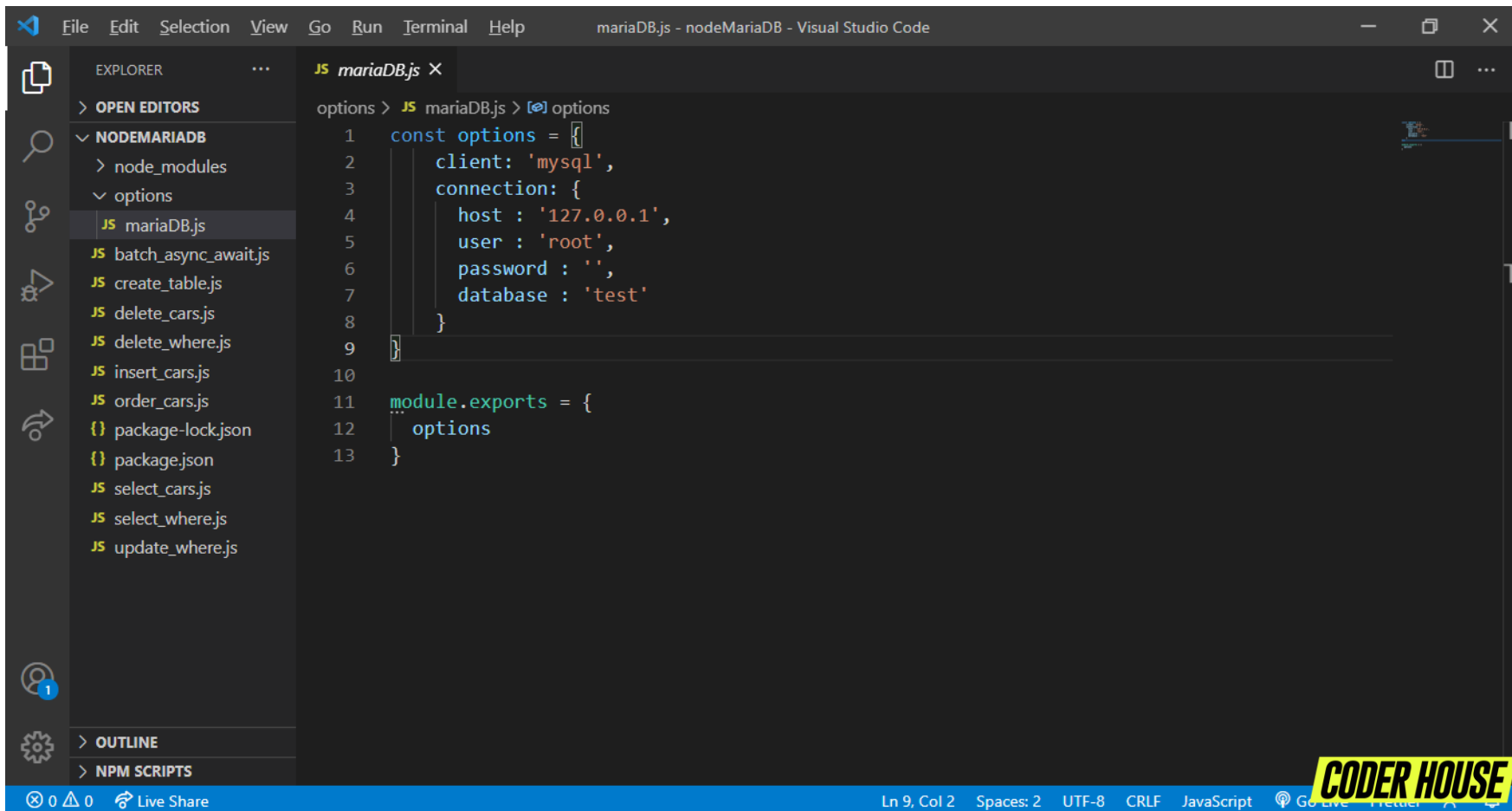
```
var pg = require('knex')({
  client: 'pg',
  connection: process.env.DATABASE_URL,
  searchPath: 'knex,public',
  pool: { min: 0, max: 7 }
})
```

Connect via Sqlite

```
var knex = require('knex')({
  client: 'sqlite3',
  connection: { filename: './mydb.sqlite' }
})
```



Proyecto Knex MariaDB: config options



The image shows a Visual Studio Code editor window with the file `mariaDB.js` open. The Explorer sidebar on the left shows the project structure, including `node_modules` and `options` folders, with `mariaDB.js` selected. The main editor area displays the following JavaScript code:

```
options > JS mariaDB.js > [e] options
1  const options = {
2      client: 'mysql',
3      connection: {
4          host : '127.0.0.1',
5          user : 'root',
6          password : '',
7          database : 'test'
8      }
9  }
10
11  module.exports = {
12      options
13  }
```

The status bar at the bottom indicates the current position is Line 9, Column 2, with 2 spaces, UTF-8 encoding, CRLF line endings, and the JavaScript language mode. A "Live Share" icon is also visible on the left of the status bar.

CODER HOUSE

Knex: Crear y Modificar tabla



Create table

```
knex.schema.createTable('accounts', table => {
```

Columns

```
table.increments('id')
table.string('account_name')
table.integer('age')
table.float('age')
table.decimal('balance', 8, 2)
table.boolean('is_admin')
table.date('birthday')
table.time('created_at')
table.timestamp('created_at').defaultTo(knex.fn.now())
table.json('profile')
table.jsonb('profile')
table.uuid('id').primary()
```

Constraints

```
table.unique('email')
table.unique(['email', 'company_id'])
table.dropUnique(...)
```

Indices

```
table.foreign('company_id')
  .references('companies.id')
table.dropForeign(...)
```

Variations

```
table.integer('user_id')
  .unsigned()
  .references('users.id')
```

```
})
.then(() => ...)
```

Alter table

```
knex.schema.table('accounts', table => {
```

Create

```
table.string('first_name')
```

Alter

```
table.string('first_name').alter()
table.renameColumn('admin', 'is_admin')
```

Drop

```
table.dropColumn('admin')
table.dropTimestamps('created_at')
```

```
})
```

See: [Schema builder](#)

Other methods

```
knex.schema
  .renameTable('persons', 'people')
  .dropTable('persons')
```

```
.hasTable('users').then(exists => ...)
.hasColumn('users', 'id').then(exists => ...)
```

See: [Schema builder](#)

Tabla 1

Campo 1	Campo 2	Campo 3	Campo 4

Tabla 2

Campo 1	Campo 2	Campo 3	Campo 4

Proyecto Knex MariaDB: create table

The screenshot displays the Visual Studio Code interface with two JavaScript files open: `create_table.js` and `mariaDB.js`.

EXPLORER

- OPEN EDITORS
- NODEMARIADB
 - node_modules
 - options
 - mariaDB.js
 - batch_async_await.js
 - create_table.js
 - delete_cars.js
 - delete_where.js
 - insert_cars.js
 - order_cars.js
 - package-lock.json
 - package.json
 - select_cars.js
 - select_where.js
 - update_where.js
- OUTLINE
- NPM SCRIPTS

create_table.js

```
1 const { options } = require('./options/mariaDB.js')
2 const knex = require('knex')(options);
3
4 //Se crea una nueva tabla con la función createTable ()
5 //del esquema Knex.js. Definimos el esquema para que
6 //contenga tres columnas: id, nombre y precio.
7 knex.schema.createTable('cars', table => {
8   table.increments('id')
9   table.string('name')
10  table.integer('price')
11 })
12 .then(() => console.log("table created"))
13 .catch((err) => { console.log(err); throw err })
14 .finally(() => {
15   knex.destroy();
16 });
```

mariaDB.js

```
1 const options = {
2   client: 'mysql',
3   connection: {
4     host: '127.0.0.1',
5     user: 'root',
6     password: '',
7     database: 'test'
8   }
9 }
10
11 module.exports = {
12   options
13 }
```

TERMINAL

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase16\nodeMariaDB> node .\create_table.js
table created
PS C:\Cursos\Coderhouse\CursoBackend\Clase16\nodeMariaDB>
```

Page-Footer

Ln 9, Col 3 Spaces: 2 UTF-8 CRLF JavaScript Go Live Prettier

CODER HOUSE

OPERATION	SQL	HTTP
Create	INSERT	PUT/POST
Read	SELECT	GET
Update	UPDATE	PUT/PATCH
Delete	DELETE	DELETE

Knex CRUD: Seleccionar, Insertar, Actualizar y Borrar datos

Insert

```
knex('users')
```

Insert one

```
.insert({ name: 'John' })
```

Insert many

```
.insert([  
  { name: 'Starsky' },  
  { name: 'Hutch' }  
])
```

See: [Insert](#)

Update

```
knex('users')  
  .where({ id: 2 })  
  .update({ name: 'Homer' })
```

See: [Update](#)

Delete

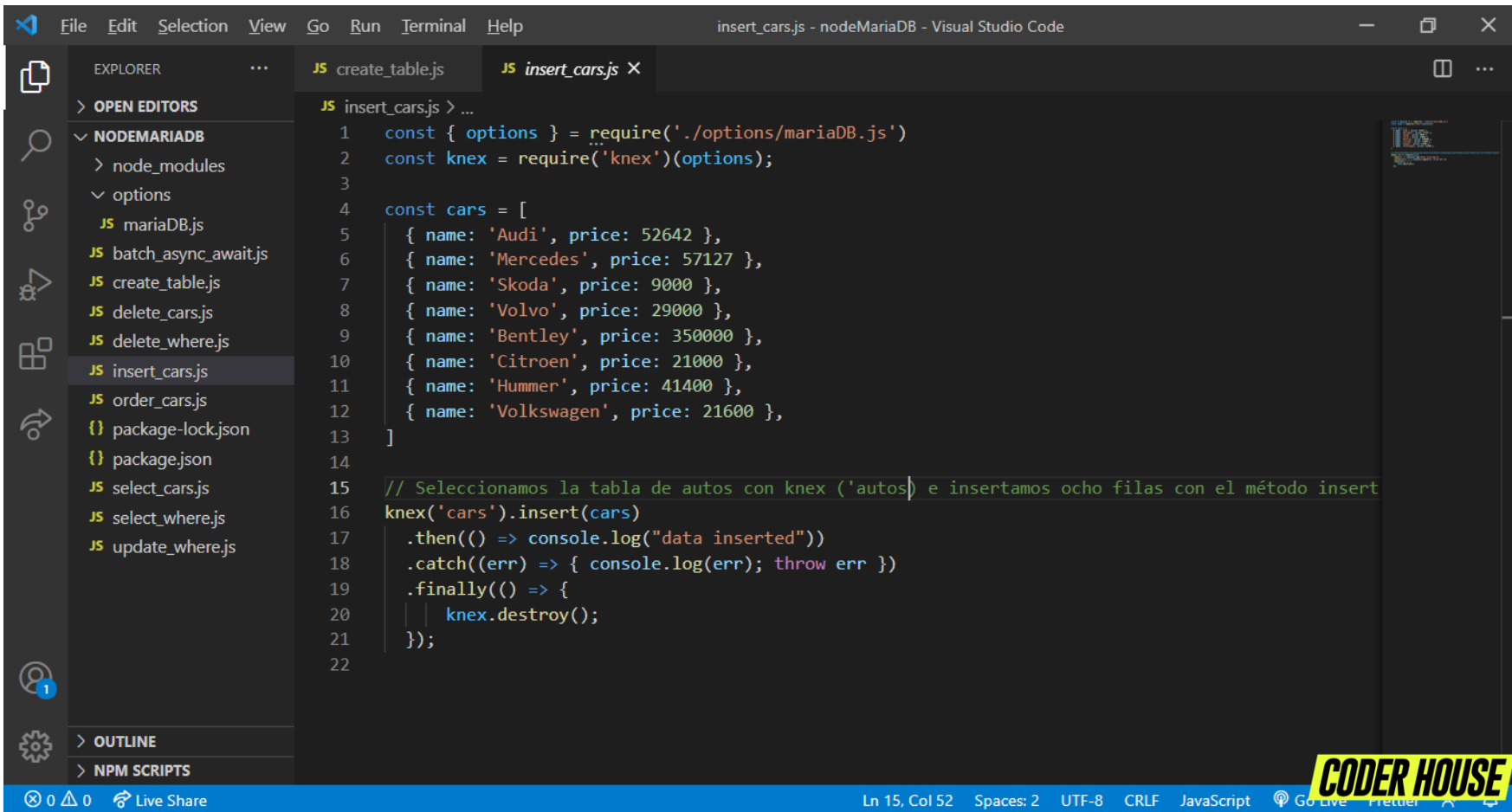
```
knex('users')  
  .where({ id: 2 })  
  .del()
```

See: [Delete](#)

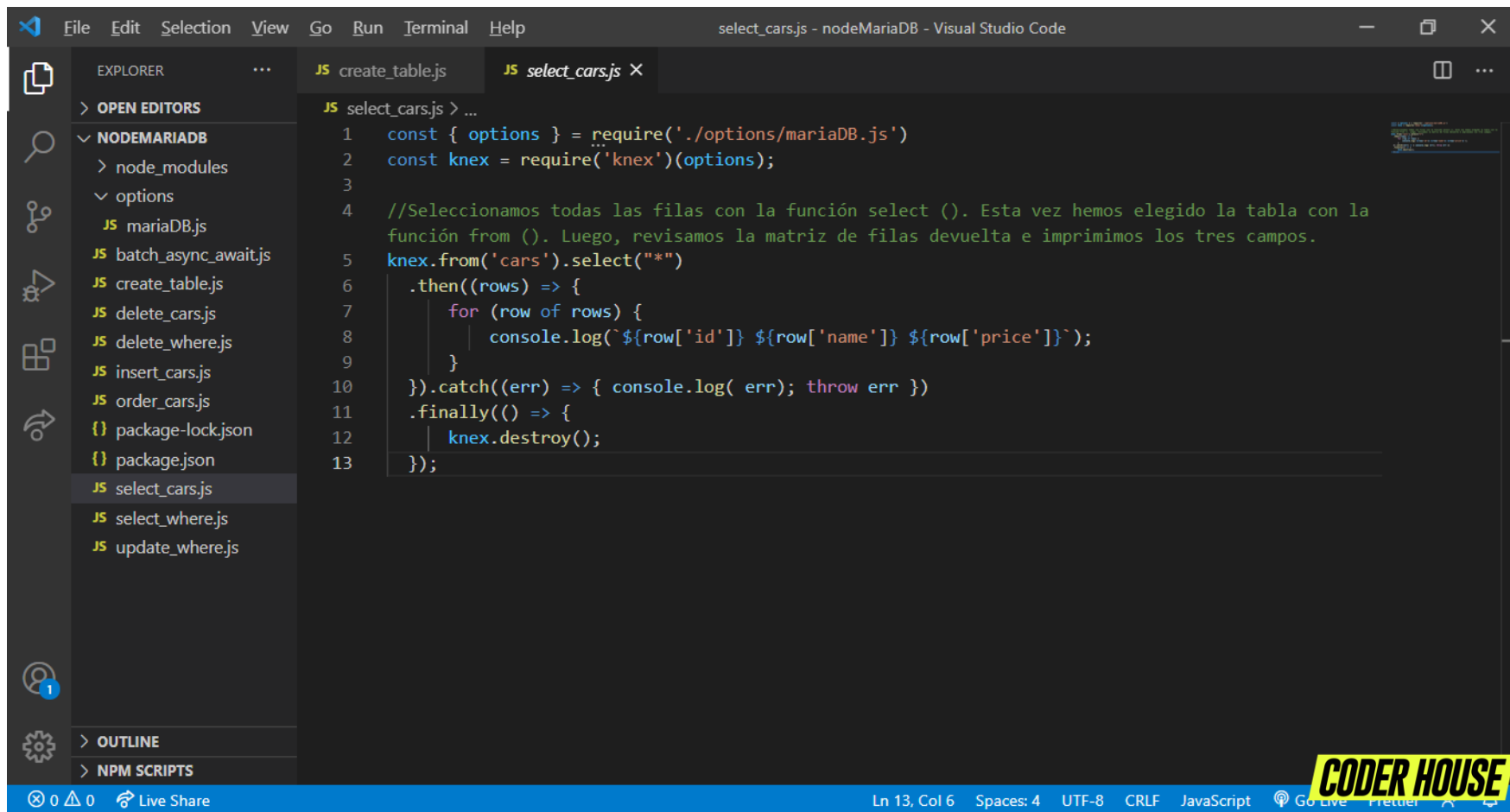
Select

```
knex  
  .from('books')  
  .select('title', 'author', 'year')
```

Proyecto Knex MariaDB: insert



Proyecto Knex MariaDB: select



select_cars.js - nodeMariaDB - Visual Studio Code

EXPLORER

- OPEN EDITORS
- NODEMARIADB
 - node_modules
 - options
 - mariaDB.js
 - batch_async_await.js
 - create_table.js
 - delete_cars.js
 - delete_where.js
 - insert_cars.js
 - order_cars.js
 - package-lock.json
 - package.json
 - select_cars.js
 - select_where.js
 - update_where.js
- OUTLINE
- NPM SCRIPTS

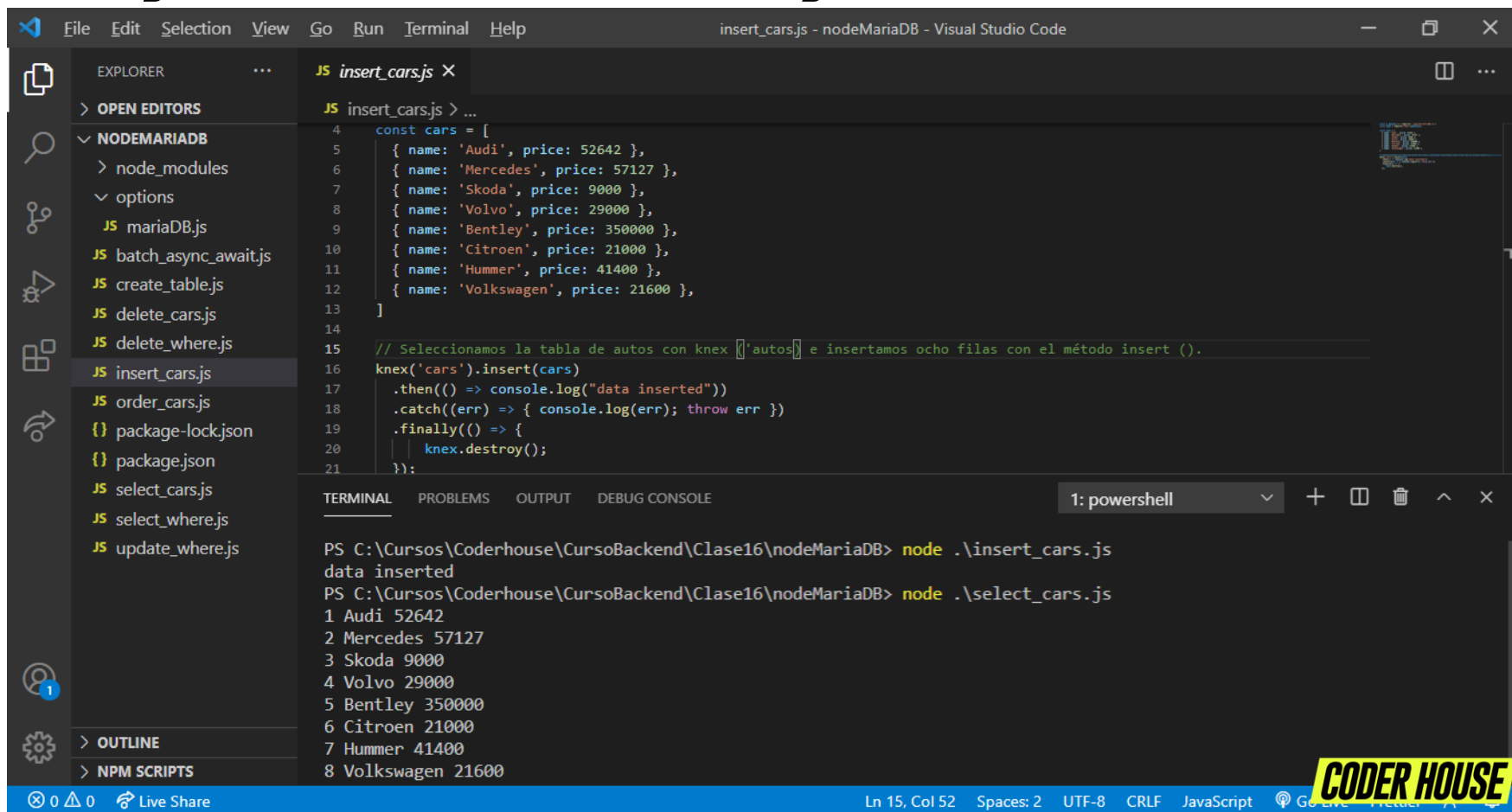
JS select_cars.js > ...

```
1 const { options } = require('./options/mariaDB.js')
2 const knex = require('knex')(options);
3
4 //Seleccionamos todas las filas con la función select (). Esta vez hemos elegido la tabla con la
  función from (). Luego, revisamos la matriz de filas devuelta e imprimimos los tres campos.
5 knex.from('cars').select("*")
6   .then((rows) => {
7     for (row of rows) {
8       console.log(`${row['id']} ${row['name']} ${row['price']}`);
9     }
10  }).catch((err) => { console.log( err); throw err })
11  .finally(() => {
12    knex.destroy();
13  });
```

Ln 13, Col 6 Spaces: 4 UTF-8 CRLF JavaScript Go Live Prettier

CODER HOUSE

Proyecto Knex MariaDB: insert y select en consola



The image shows a Visual Studio Code editor window with the file `insert_cars.js` open. The Explorer sidebar on the left shows the project structure, including `node_modules`, `options`, and several JavaScript files. The `insert_cars.js` file is selected, showing the following code:

```
JS insert_cars.js > ...
4   const cars = [
5     { name: 'Audi', price: 52642 },
6     { name: 'Mercedes', price: 57127 },
7     { name: 'Skoda', price: 9000 },
8     { name: 'Volvo', price: 29000 },
9     { name: 'Bentley', price: 350000 },
10    { name: 'Citroen', price: 21000 },
11    { name: 'Hummer', price: 41400 },
12    { name: 'Volkswagen', price: 21600 },
13  ]
14
15  // Seleccionamos la tabla de autos con knex ['autos'] e insertamos ocho filas con el método insert ().
16  knex('cars').insert(cars)
17    .then(() => console.log("data inserted"))
18    .catch((err) => { console.log(err); throw err })
19    .finally(() => {
20      |   knex.destroy();
21    });
```

The TERMINAL panel at the bottom shows the execution of the script using Node.js. The output displays the data inserted into the database:

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase16\nodeMariaDB> node .\insert_cars.js
data inserted
PS C:\Cursos\Coderhouse\CursoBackend\Clase16\nodeMariaDB> node .\select_cars.js
1 Audi 52642
2 Mercedes 57127
3 Skoda 9000
4 Volvo 29000
5 Bentley 350000
6 Citroen 21000
7 Hummer 41400
8 Volkswagen 21600
```

The bottom status bar indicates the current position is Ln 15, Col 52, with 2 spaces, UTF-8 encoding, CRLF line endings, and the file is a JavaScript document.

CODER HOUSE



BREAK

¡5/10 MINUTOS Y VOLVEMOS!



Knex: Where y otros comandos

Where

```
.where('title', 'Hello')  
.where({ title: 'Hello' })  
.whereIn('id', [1, 2, 3])  
.whereNot(...)  
.whereNotIn('id', [1, 2, 3])
```

Where conditions

```
.whereNull('updated_at')  
.whereNotNull(...)
```

```
.whereExists('updated_at')  
.whereNotExists(...)
```

```
.whereBetween('votes', [1, 100])  
.whereNotBetween(...)
```

```
.whereRaw('id = ?', [1])
```

Where grouping

```
.where(function () {  
  this  
    .where('id', 1)  
    .orWhere('id', '>', 10)  
})
```

Others

```
knex('users')  
  .distinct()
```

Group

```
.groupBy('count')  
.groupByRaw('year WITH ROLLUP')
```

Order

```
.orderBy('name', 'desc')  
.orderByRaw('name DESC')
```

Offset/limit

```
.offset(10)  
.limit(20)
```

Having

```
.having('count', '>', 100)  
.havingIn('count', [1, 100])
```

Union

```
.union(function() {  
  this.select(...)  
})  
.unionAll(...)
```

Proyecto Knex MariaDB: select where

select_where.js - nodeMariaDB - Visual Studio Code

EXPLORER

OPEN EDITORS

NODEMARIADB

- node_modules
- options
 - mariaDB.js
- batch_async_await.js
- create_table.js
- delete_cars.js
- delete_where.js
- insert_cars.js
- order_cars.js
- package-lock.json
- package.json
- select_cars.js
- select_where.js
- update_where.js

OUTLINE

NPM SCRIPTS

```
JS select_where.js > ...
1  const { options } = require('./options/mariaDB.js')
2  const knex = require('knex')(options);
3
4  //El ejemplo devuelve coches cuyo precio es superior a 50000.
5  knex.from('cars').select("name", "price").where('price', '>', '50000')
6    .then((rows) => {
7      for (row of rows) {
8        console.log(`${row['name']} ${row['price']}`);
9      }
10   })
11   .catch((err) => { console.log( err); throw err })
12   .finally(() => {
13     knex.destroy();
14   });
```

Ln 5, Col 49 Spaces: 4 UTF-8 CRLF JavaScript

CODER HOUSE

Proyecto Knex MariaDB: order by

order_cars.js - nodeMariaDB - Visual Studio Code

```
JS order_cars.js > ...
1  const { options } = require('./options/mariaDB.js')
2  const knex = require('knex')(options);
3
4  //El ejemplo selecciona todos los coches y los ordena por precio en orden descendente.
5  knex.from('cars').select('name', 'price').orderBy('price', 'desc')
6    .then((rows) => {
7      for (row of rows) {
8        console.log(`${row['name']} ${row['price']}`);
9      }
10   }).catch((err) => { console.log( err); throw err })
11   .finally(() => {
12     knex.destroy();
13   });
```

Ln 13, Col 8 Spaces: 4 UTF-8 CRLF JavaScript

CODER HOUSE

Proyecto Knex MariaDB: update

update_where.js - nodeMariaDB - Visual Studio Code

EXPLORER

OPEN EDITORS

NODEMARIADB

- node_modules
- options
 - mariaDB.js
- batch_async_await.js
- create_table.js
- delete_cars.js
- delete_where.js
- insert_cars.js
- order_cars.js
- package-lock.json
- package.json
- select_cars.js
- select_where.js
- update_where.js

OUTLINE

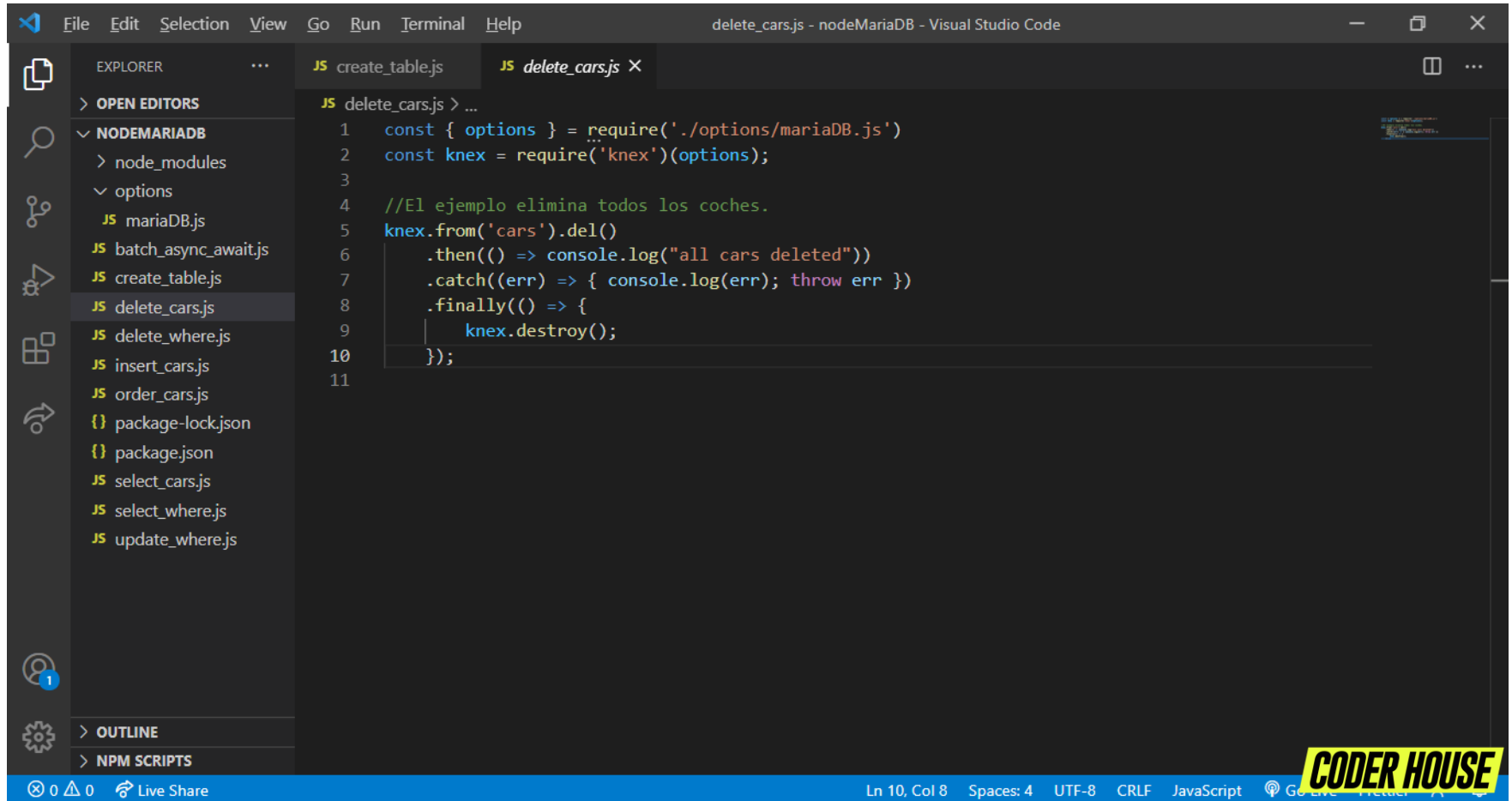
NPM SCRIPTS

```
JS update_where.js > ...
1  const { options } = require('./options/mariaDB.js')
2  const knex = require('knex')(options);
3
4  //El ejemplo actualiza coches cuyo precio es igual a 9000 con 9500
5  knex.from('cars').where('price', '9000').update({price: 9500})
6    .then(() => console.log("car updated"))
7    .catch((err) => { console.log(err); throw err })
8    .finally(() => {
9      |   knex.destroy();
10  });|
11
```

Ln 10, Col 8 Spaces: 4 UTF-8 CRLF JavaScript

CODER HOUSE

Proyecto Knex MariaDB: delete



The screenshot shows the Visual Studio Code interface with the following details:

- Explorer Panel:** Displays the file structure of the 'nodeMariaDB' project. The 'delete_cars.js' file is selected and highlighted.
- Editor Panel:** Shows the content of 'delete_cars.js' with the following code:

```
1 const { options } = require('./options/mariaDB.js')
2 const knex = require('knex')(options);
3
4 //El ejemplo elimina todos los coches.
5 knex.from('cars').del()
6   .then(() => console.log("all cars deleted"))
7   .catch((err) => { console.log(err); throw err })
8   .finally(() => {
9     |   knex.destroy();
10  });
11
```
- Footer:** The status bar at the bottom indicates 'Ln 10, Col 8', 'Spaces: 4', 'UTF-8', 'CRLF', 'JavaScript', and a 'Live Share' button. A 'CODER HOUSE' watermark is visible in the bottom right corner.

Proyecto Knex MariaDB: delete where

delete_where.js - nodeMariaDB - Visual Studio Code

EXPLORER

OPEN EDITORS

NODEMARIADB

- node_modules
- options
 - JS mariaDB.js
 - JS batch_async_await.js
 - JS create_table.js
 - JS delete_cars.js
 - JS delete_where.js
 - JS insert_cars.js
 - JS order_cars.js
- package-lock.json
- package.json
- JS select_cars.js
- JS select_where.js
- JS update_where.js

OUTLINE

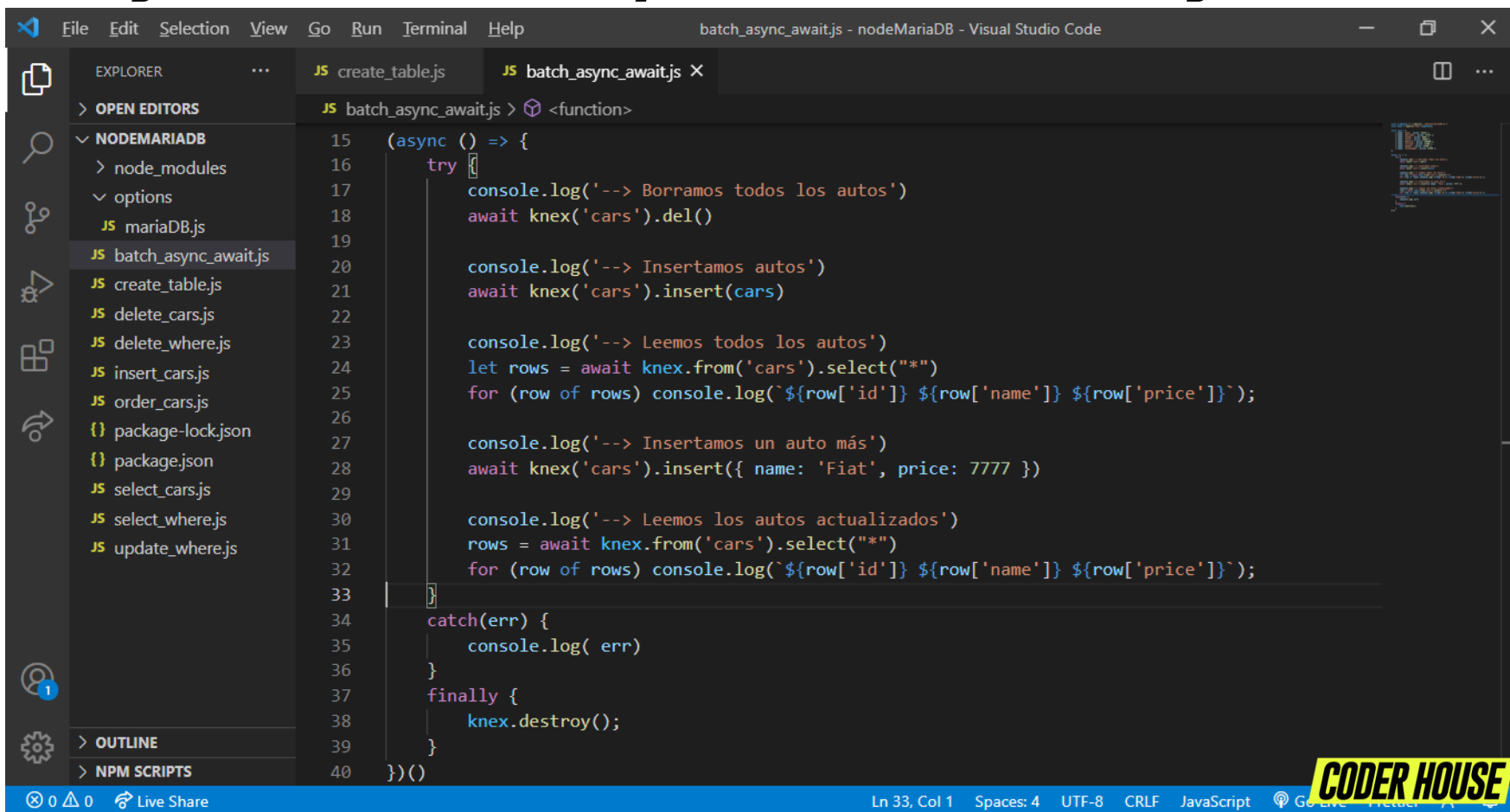
NPM SCRIPTS

```
JS delete_where.js > ...
1  const { options } = require('./options/mariaDB.js')
2  const knex = require('knex')(options);
3
4  //El ejemplo elimina coches cuyo precio es superior a 50000.
5  knex.from('cars').where('price', '>', '50000').del()
6    .then(() => console.log("cars deleted"))
7    .catch((err) => { console.log(err); throw err })
8    .finally(() => {
9      knex.destroy();
10    });
11
```

Ln 11, Col 1 Spaces: 4 UTF-8 CRLF JavaScript

CODER HOUSE

Proyecto Knex MariaDB: procesos batch con async await



batch_async_await.js - nodeMariaDB - Visual Studio Code

```
15 (async () => {
16   try {
17     console.log('--> Borramos todos los autos')
18     await knex('cars').del()
19
20     console.log('--> Insertamos autos')
21     await knex('cars').insert(cars)
22
23     console.log('--> Leemos todos los autos')
24     let rows = await knex.from('cars').select("*")
25     for (row of rows) console.log(`${row['id']} ${row['name']} ${row['price']}`);
26
27     console.log('--> Insertamos un auto más')
28     await knex('cars').insert({ name: 'Fiat', price: 7777 })
29
30     console.log('--> Leemos los autos actualizados')
31     rows = await knex.from('cars').select("*")
32     for (row of rows) console.log(`${row['id']} ${row['name']} ${row['price']}`);
33   }
34   catch(err) {
35     console.log( err)
36   }
37   finally {
38     knex.destroy();
39   }
40 })()
```

0 0 0 Live Share Ln 33, Col 1 Spaces: 4 UTF-8 CRLF JavaScript

CODER HOUSE



NODE + MARIADB

Tiempo: 15 minutos



Realizar un proyecto en Node.js que se conecte a la base de datos llamada *ecommerce* implementada en MariaDB y ejecute las siguientes procesos:

1. Debe crear una tabla llamada *articulos* con la siguiente estructura:

Campos:

- **nombre** tipo varchar 15 caracteres no nulo
 - **codigo** tipo varchar 10 caracteres no nulo
 - **precio** tipo float
 - **stock** tipo entero
 - **id clave primaria** autoincremental no nula
1. Insertar 5 articulos en esa tabla, con datos de prueba con stocks positivos
 2. Listar la tabla mostrando los resultados en la consola
 3. Borrar el articulo con id = 3
 4. Actualizar el stock a 0 del articulo con id = 2



Notas:

- Crear un único archivo ejecutable a través de node.js que realice lo pedido. Considerar que estos son procesos asincrónicos que devuelven promesas y deben ser anidados para mantener el orden de operación. Utilizar la sintaxis then/catch
- Agregar como primera acción que, en caso de existir la tabla, la borre (drop), así al ejecutar estas mismas tareas, empezamos desde cero sin errores y datos residuales.

Node.js como cliente de SQLite3





¿Qué es SQLite3?



- SQLite es una **biblioteca en lenguaje C** que implementa un motor de base de datos SQL pequeño, rápido, autónomo, de alta confiabilidad y con todas las funciones.
- SQLite es el **motor de base de datos más utilizado** del mundo.
- SQLite está **integrado** en todos los **teléfonos móviles** y en la mayoría de las **computadoras** y viene incluido dentro de innumerables otras aplicaciones que la gente usa todos los días.
- El **formato de archivo** SQLite es **estable, multiplataforma** y **compatible** con versiones anteriores. La última versión es la 3
- El **código fuente** de SQLite es de **dominio público**.

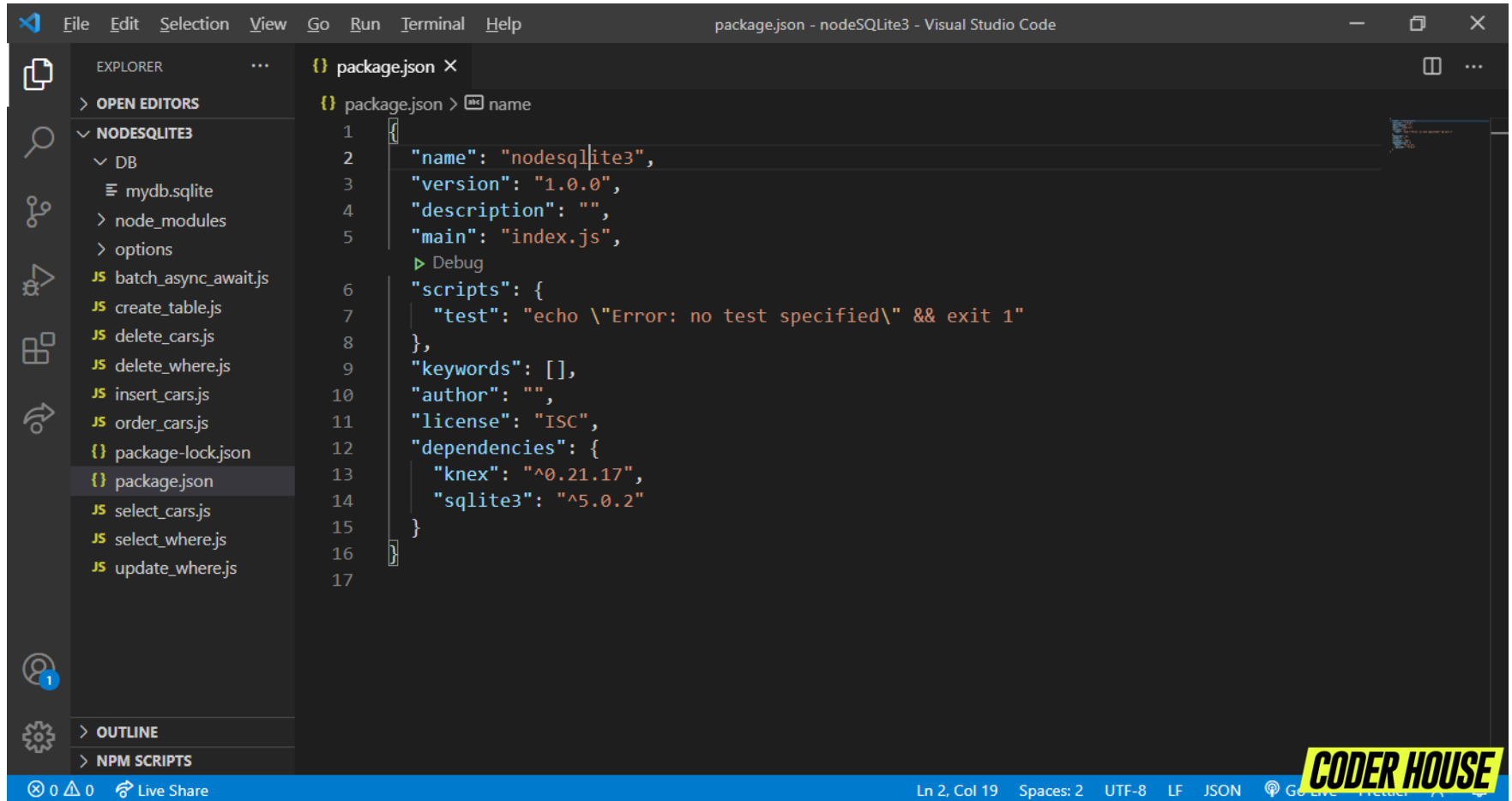
Proyecto Cliente SQLite3 Node.js con Knex.js



Inicialización del proyecto e instalación de dependencias

1. Creamos un proyecto Node.js con **npm init -y**
2. Instalamos la dependencias *Knex* y *sqlite3* con **npm i knex
sqlite3** (**sqlite3** es el plugin necesario para trabajar con *SQLite
Version 3*)
3. Creamos la carpeta DB, en donde se va a almacenar el archivo que contiene la estructura de base de datos.
4. Creamos los archivos necesarios para probar los comandos SQL necesarios en acciones CRUD.

Proyecto Knex SQLite3: package.json



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the project structure, including a folder named 'NODESQLITE3' containing a 'DB' subfolder with 'mydb.sqlite', and several JavaScript files. The 'package.json' file is selected and open in the main editor. The package.json content is as follows:

```
{
  "name": "nodesqlite3",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "knex": "^0.21.17",
    "sqlite3": "^5.0.2"
  }
}
```

The status bar at the bottom indicates the current position is Line 2, Column 19, with 2 spaces, UTF-8 encoding, and LF line endings. A 'Live Share' icon is visible on the left, and a 'CODER HOUSE' watermark is present in the bottom right corner.

Knex: Conexión a SQLite3



Connect via Sqlite

```
var knex = require('knex')({  
  client: 'sqlite3',  
  connection: { filename: './mydb.sqlite' }  
})
```

El resto del proyecto es igual al realizado anteriormente con MariaDB. Sólo cambia la base de trabajo, manteniendo las funciones, sus llamados y devoluciones de forma similar.

Proyecto Knex SQLite3: config options

Visual Studio Code interface showing the configuration of Knex.js for SQLite3. The Explorer sidebar on the left shows the project structure, including files like `mydb.sqlite`, `node_modules`, and `options`. The main editor window displays the `SQLite3.js` file with the following code:

```
options > JS SQLite3.js > [?] <unknown>
1  const options = {
2    client: 'sqlite3',
3    connection: {
4      filename: './DB/mydb.sqlite'
5    },
6    useNullAsDefault: true
7  }
8
9  module.exports = {
10   options
11 }
```

The status bar at the bottom indicates the current line and column (Ln 11, Col 2), spaces (Spaces: 2), encoding (UTF-8), line endings (CRLF), and the language (JavaScript). A "Go Live" button is also visible. A "CODER HOUSE" watermark is present in the bottom right corner.



NODE + SQLITE3

Tiempo: 15 minutos



Realizar un proyecto en Node.js que se conecte a una base de datos SQLite3 y ejecute las mismas acciones que las planteadas en el desafío anterior.

Notas:

- Crear un único archivo ejecutable a través de node.js que realice lo pedido. Considerar que estos son procesos asincrónicos que devuelven promesas y deben ser anidados para mantener el orden de operación. Utilizar la sintaxis async/await.
- Agregar como primera acción que si existe la tabla la borre (drop), así, al ejecutar estas mismas tareas, comienzo de cero sin errores y datos residuales.



NUESTRA PRIMERA BASE DE DATOS

Nuestra Primera Base de Datos

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Desafío
entregable



>> Consigna: Tomando como base las clases Contenedor en memoria y en archivos, desarrollar un nuevo contenedor con idénticos métodos pero que funcione sobre bases de datos, utilizando Knex para la conexión. Esta clase debe recibir en su constructor el objeto de configuración de Knex y el nombre de la tabla sobre la cual trabajará. Luego, modificar el desafío entregable de la clase 11 "Chat con Websocket", y:

- cambiar la persistencia de los mensajes de filesystem a base de datos SQLite3.
- cambiar la persistencia de los productos de memoria a base de datos MariaDB.

Desarrollar también un script que utilizando knex cree las tablas necesarias para la persistencia en cuestión (tabla mensajes en sqlite3 y tabla productos en mariaDb).

>> Notas:

- Definir una carpeta DB para almacenar la base datos SQLite3 llamada ecommerce

CODER HOUSE

¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- MySQL.
 - MariaDB.
 - SQLite3.
 - Knex.JS.
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN