



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



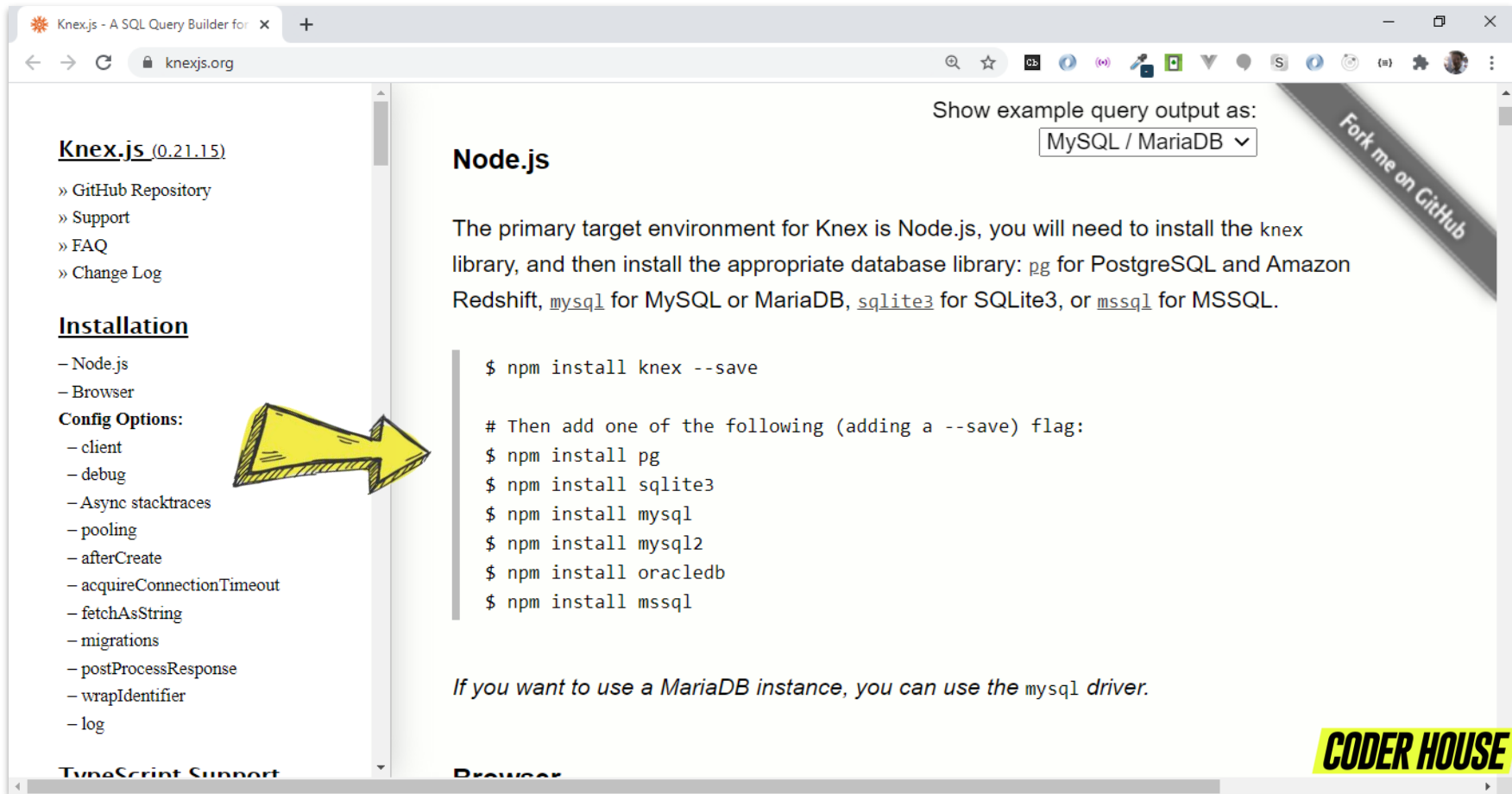
KNEX.JS

¿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



The screenshot shows the Knex.js website in a web browser. The left sidebar contains a navigation menu with links to the GitHub Repository, Support, FAQ, and Change Log. Below these are sections for 'Installation' and 'Config Options'. A large yellow arrow points from the 'Installation' section to the main content area. The main content area has a heading 'Node.js' and a paragraph explaining that Node.js is the primary target environment. It lists the steps to install Knex and the appropriate database driver (pg, mysql, sqlite3, or mssql). A code block shows the terminal commands for installation. A dropdown menu allows selecting the database type, currently set to 'MySQL / MariaDB'. A diagonal banner on the right says 'Fork me on GitHub'. The bottom right corner features a 'CODER HOUSE' logo.

Knex.js - A SQL Query Builder for [Node.js](#)

knexjs.org

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.

```
$ 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
```

Show example query output as: MySQL / MariaDB

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

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/  
})
```

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

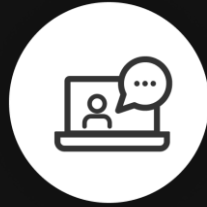
```
knex seed:make seed_name  
knex seed:make seed_name -x ts # Generates a TypeScript seed file  
knex seed:run # Runs all seed files  
knex seed:run --specific=seed-filename.js # Run a specific seed file
```

See: [Seeds](#)



***¿Alguna pregunta hasta
ahora?***

Vamos al IDE... 



***¿Alguna pregunta hasta
ahora?***



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(...)
```

Vamos al IDE... 



***¿Alguna pregunta hasta
ahora?***



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:

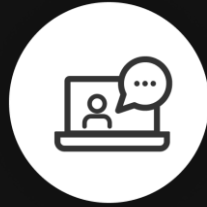
Campos:

- **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.



***¿Alguna pregunta hasta
ahora?***

Node.js como cliente de SQLite3



Vamos al IDE... 



¿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**.



***¿Alguna pregunta hasta
ahora?***



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.



***¿Alguna pregunta hasta
ahora?***



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*

¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

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



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN