**Getting Started**

In this tutorial you will learn to make a simple setup of Sequelize.

**Installing**[**​**](https://sequelize.org/docs/v6/getting-started/#installing)

Sequelize is available via [npm](https://www.npmjs.com/package/sequelize) (or [yarn](https://yarnpkg.com/package/sequelize)).

npm install --save sequelize

You'll also have to manually install the driver for your database of choice:

# One of the following:  
$ npm install --save pg pg-hstore # Postgres  
$ npm install --save mysql2  
$ npm install --save mariadb  
$ npm install --save sqlite3  
$ npm install --save tedious # Microsoft SQL Server  
$ npm install --save oracledb # Oracle Database

**Connecting to a database**[**​**](https://sequelize.org/docs/v6/getting-started/#connecting-to-a-database)

To connect to the database, you must create a Sequelize instance. This can be done by either passing the connection parameters separately to the Sequelize constructor or by passing a single connection URI:

const { Sequelize } = require('sequelize');  
  
*// 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' | 'postgres' | 'sqlite' | 'mariadb' | 'mssql' | 'db2' | 'snowflake' | 'oracle' \*/*  
});

The Sequelize constructor accepts a lot of options. They are documented in the [API Reference](https://sequelize.org/api/v6/class/src/sequelize.js~sequelize#instance-constructor-constructor).

**Testing the connection**[**​**](https://sequelize.org/docs/v6/getting-started/#testing-the-connection)

You can use the .authenticate() function to test if the connection is OK:

try {  
 await sequelize.authenticate();  
 console.log('Connection has been established successfully.');  
} catch (error) {  
 console.error('Unable to connect to the database:', error);  
}

**Closing the connection**[**​**](https://sequelize.org/docs/v6/getting-started/#closing-the-connection)

Sequelize will keep the connection open by default, and use the same connection for all queries. If you need to close the connection, call sequelize.close() (which is asynchronous and returns a Promise).

NOTE

Once sequelize.close() has been called, it's impossible to open a new connection. You will need to create a new Sequelize instance to access your database again.

**Terminology convention**[**​**](https://sequelize.org/docs/v6/getting-started/#terminology-convention)

Observe that, in the examples above, Sequelize refers to the library itself while sequelize refers to an instance of Sequelize, which represents a connection to one database. This is the recommended convention and it will be followed throughout the documentation.

**Tip for reading the docs**[**​**](https://sequelize.org/docs/v6/getting-started/#tip-for-reading-the-docs)

You are encouraged to run code examples locally while reading the Sequelize docs. This will help you learn faster. The easiest way to do this is using the SQLite dialect:

const { Sequelize, Op, Model, DataTypes } = require("sequelize");  
const sequelize = new Sequelize("sqlite::memory:");  
  
*// Code here! It works!*

To experiment with the other dialects, which are harder to setup locally, you can use the [Sequelize SSCCE](https://github.com/papb/sequelize-sscce) GitHub repository, which allows you to run code on all supported dialects directly from GitHub, for free, without any setup!