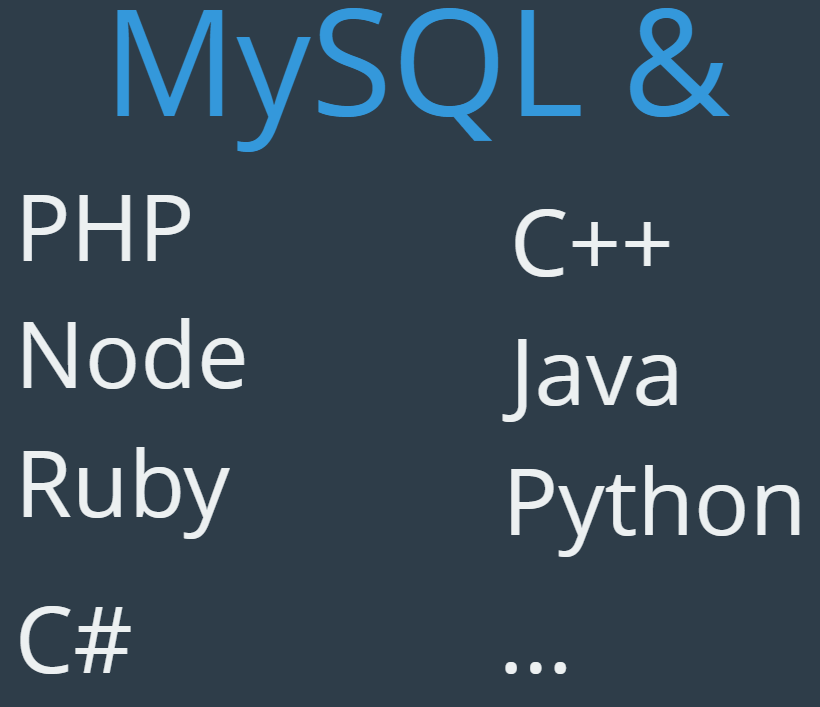
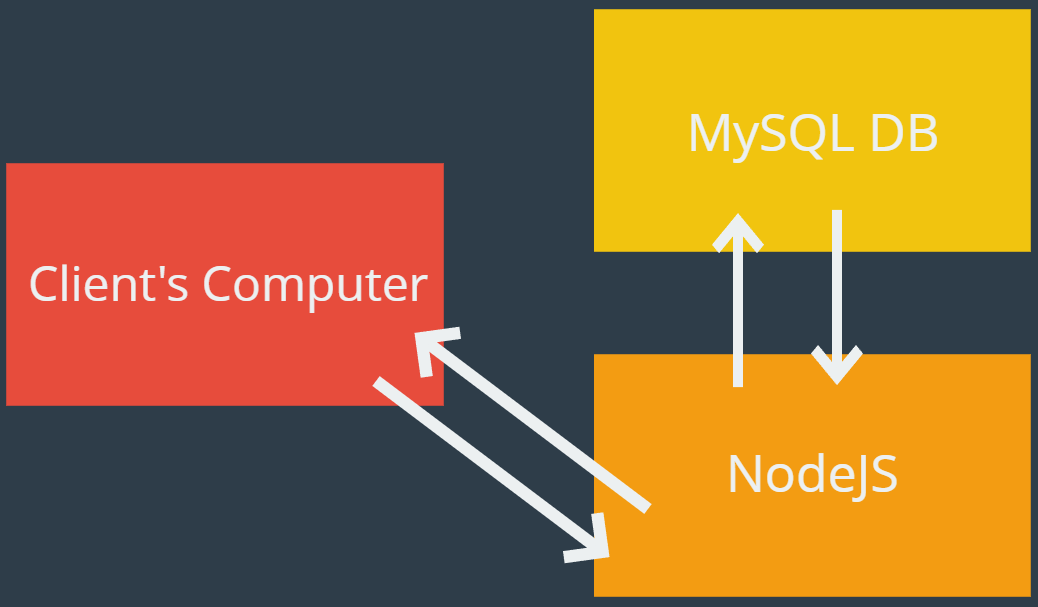
* Section slides: <http://webdev.slides.com/coltsteele/mysql-105#/>
* In this section, we’re going to see how to connect MySQL to another language, in this case node.js.
* We will connect a JavaScript file and connect it to a MySQL database

# MySQL and Other Languages

* So far we’ve been working with MySQL on its own. Now we’re going to lay the foundations for creating web applications with MySQL
  + We’ll learn how to get an external language to talk to MySQL
* We’ve been working with our CLI to ask questions of our data using MySQL, and that is a very common and important use of MySQL
* But now it’s time for this:
  + All of these languages have ways of communicating with MySQL



* + In this class we will work with Node, which is a JavaScript runtime environment that can execute JavaScript code outside of a web browser
    - https://en.wikipedia.org/wiki/Node.js
  + We could have gone with PHP, as it’s historically been closely associated with MySQL. The next lecture will talk about what might be going on with PHP
  + So, how do we talk to MySQL through external code, and what would it do? Check out this schematic
    - A client’s computer (via a web app) makes a request to NodeJS (a website’s stack).
    - Then the stack (e.g. Node) will talk to the MySQL database, creating a query depending on what the client wants
    - Node then will return the queried items to NodeJS (or whatever language), which will then compile the result, build a response (e.g. a webpage), and shoots it back to the client

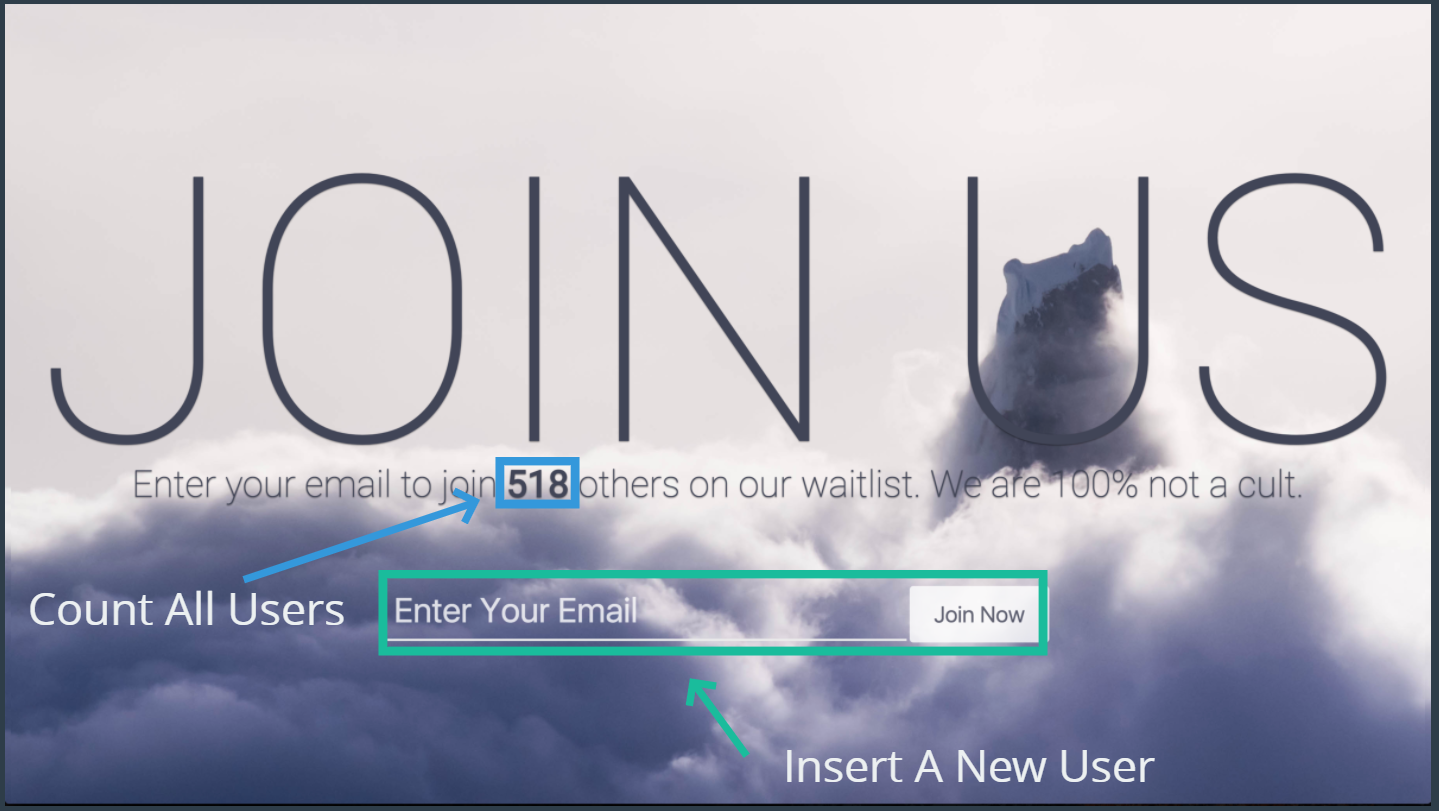


# So Why Don’t We Use PHP?

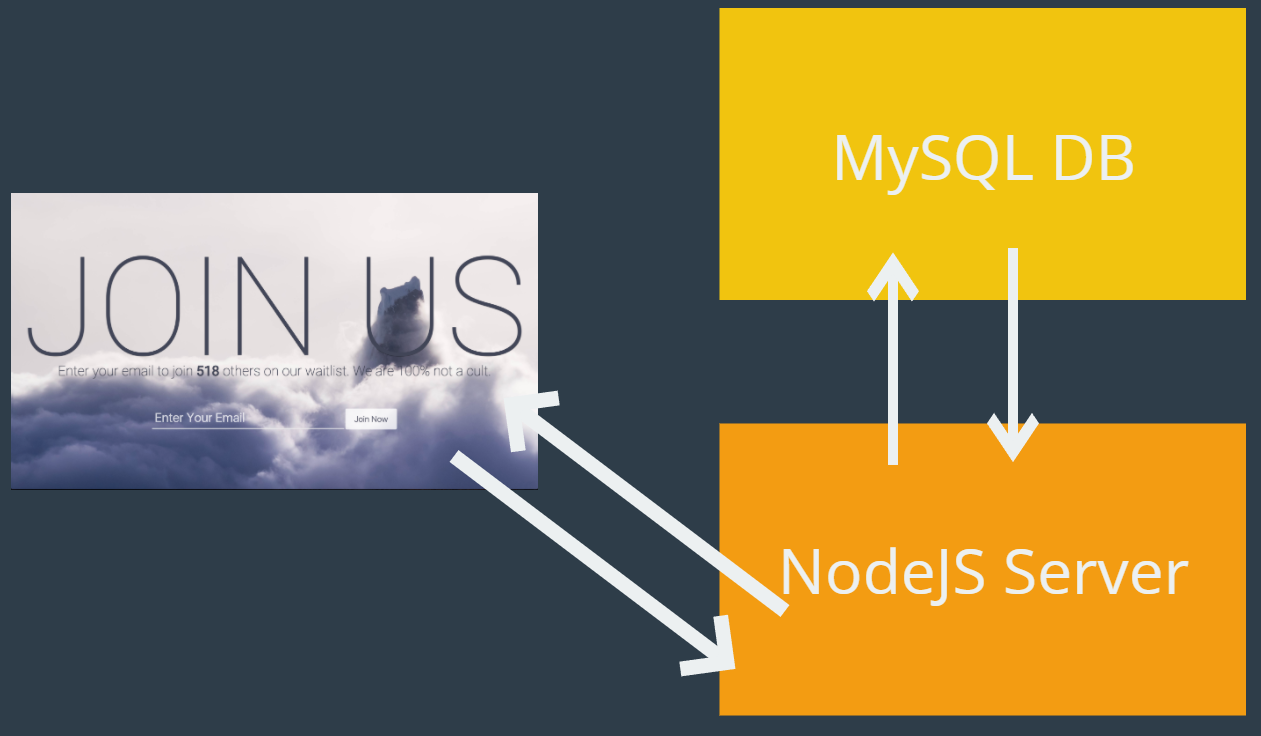
* People are oftentimes overwhelmed when trying to learn programming languages
  + Is it better to learn the easiest thing, or the most popular thing?
* PHP would be the conventional choice for, but it is no longer the most widely used language for web apps
  + People have shifted to Node, Ruby, and others
* So why do people hate PHP now?
  + It wasn’t a consciously-crafted programming language. It was pieced together over the years
  + Has security and consistency issues
    - Can also be said of JavaScript, by the way
  + The 2016 Developer Survey revealed that the two most popular technologies were JavaScript and SQL. PHP dropped to 25.9% usage by 2016. In the same time, usage of Node js has increased
* This doesn’t mean you should choose a language that’s trendy. But it would be helpful to know it for the sake of usefulness
* JavaScript and SQL are highly correlated technologies

# The JOIN US App

* We’re going to build a simple web app that uses Node and MySQL together. It is a startup mailing list application
  + This would be an app used by new companies to help them get interested parties registered and logged into their system
* This app will showcase selecting information and inserting information. The main functions will be to:
  + Count all users in the database
  + Insert a new user into the database



* The workflow
  + When a user goes to the webpage, a request is made to the Node js server to go to the main page
  + The Node js server recognizes that the main page needs to display the total number of users the database
  + Node js then goes to the MySQL database and counts the users, returning that number to the Node js server
  + Finally, the Node js server plugs the number into the page and serves the page to the requester
  + A similar workflow occurs when adding an email address



* We will start at the backend, connecting the MySQL database to the NodeJS server
  + We’ll learn how to use NodeJS to communicate with MySQL and perform MySQL activities
* We will only have one table that stores a user’s email and a timestamp of when it was created
  + Our first goal is to use NodeJS to randomly generate and insert 500+ users into a database. This will demonstrate the power of using a language like node to quickly generate data and insert it into MySQL tables

# Setting Up Node JS with Goorm

* <https://gist.github.com/nax3t/73b9cd284cae96c05b1a0d50405e753f>

# 5 Minute Crash Course on NodeJS

* Writing and executing code

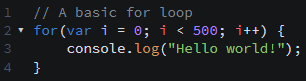


* Executing a console.log() print





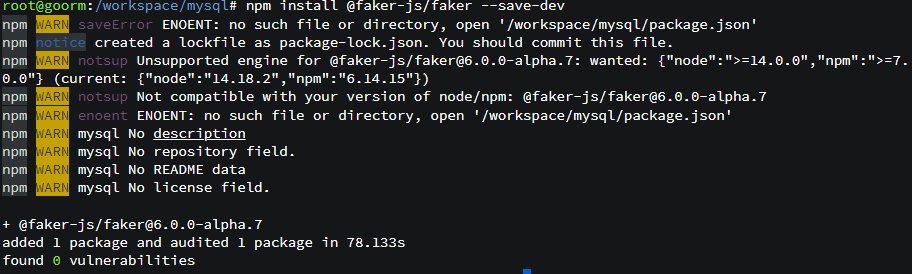
* Example of something that’s actually useful: a loop!



* + This prints “Hello world!” 500 times. This will be useful for things like inserting many users at once, instead of doing it manually
* Differences between **var**, **let**, and **const**: <https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/>
* What is the difference between NodeJS and JavaScript?
  + JavaScript is a language that was created first, and it can be used on the **client side**.
    - That means you can write code that does something on your own computer
    - Traditionally used on the front end to decorate pages, not to access databases
  + NodeJS is an implementation of JS so you can use it on the backend as a server-side language
    - Create servers
    - Talk to databases

# NPM and Faker

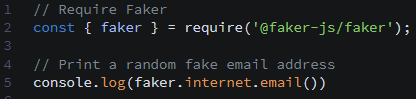
* The syntax for updating Faker has been updated from the outdated syntax that Colt uses in the course video. Updated instructions here: <https://www.loom.com/share/7ac84c0290d3489a9f43ba4fd0a47f75>
  + Run this command in the terminal to install the package:
    - npm install @faker-js/faker --save-dev
  + Enter this code in your file to require it:
    - const { faker } = require('@faker-js/faker');
* **Faker** is a node *package* that someone else has written and can very easily included in our application
  + There are Faker implementations for multiple languages
  + It streamlines the process for generating fake data. We’ll need it to create fake names and emails for our database
  + It can generate fake names, phone numbers, addresses, credit card numbers, etc.
* Let’s install Faker using **npm**



* In order to use Faker in any of our JavaScript files, we need to *require* it, which we can do with this code

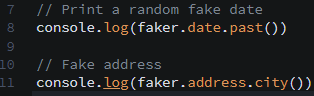


* Now we can use it. Let’s try creating some fake user info
  + Fake email



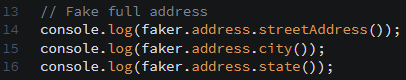


* + Fake date and city



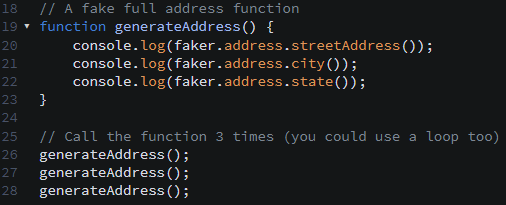


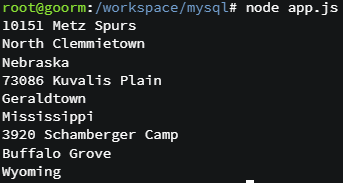
* Let’s string a few things together with some more complex logic. Here we’ll create a complete address with a number, street, and city





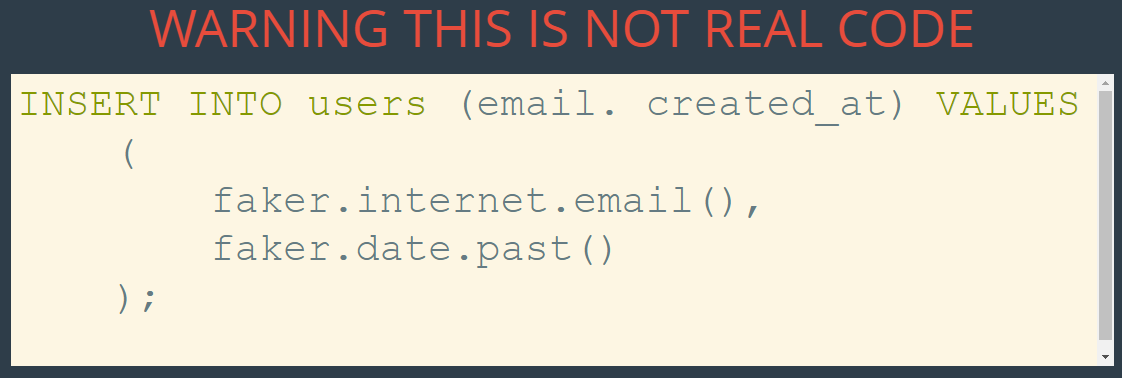
* Now if we want to do this programmatically to do it over and over again, we can write a **function** and simply call it as many times as we want!



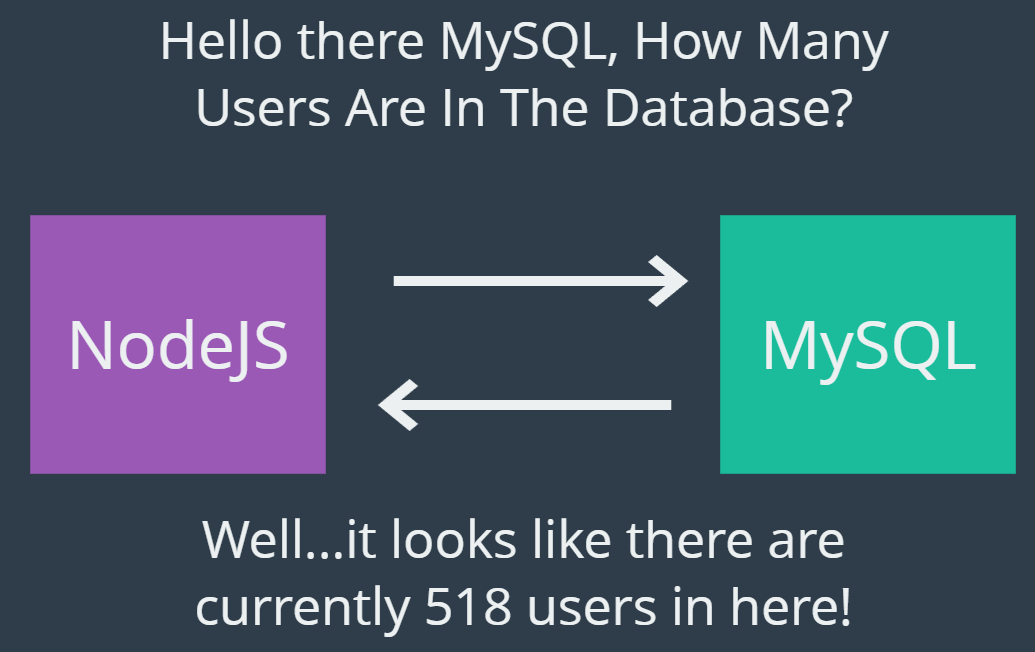


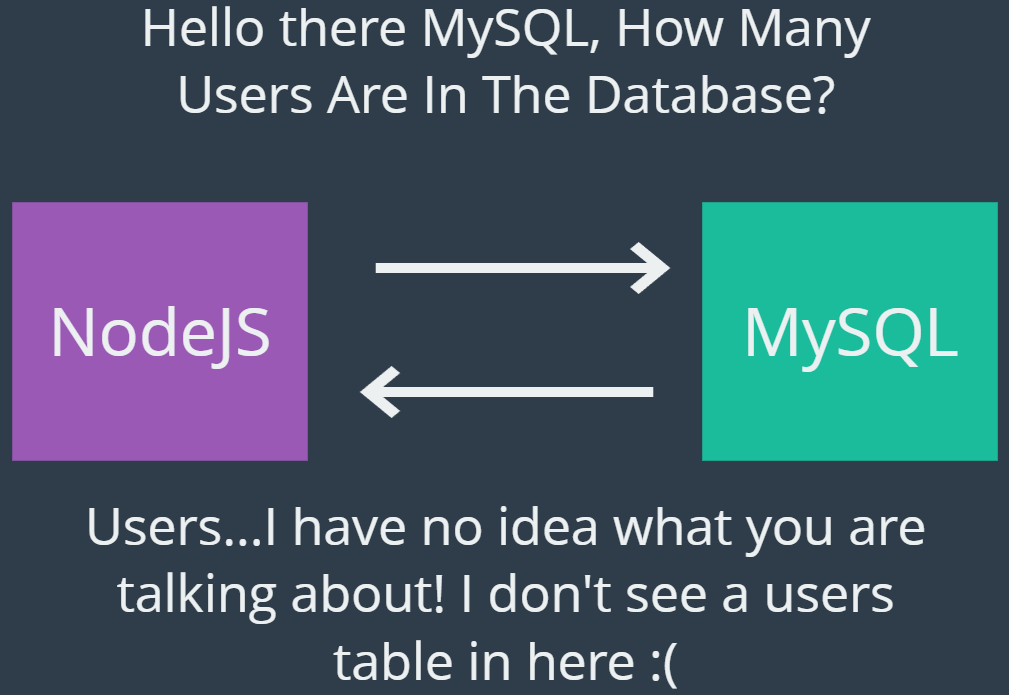
# The MySQL Package

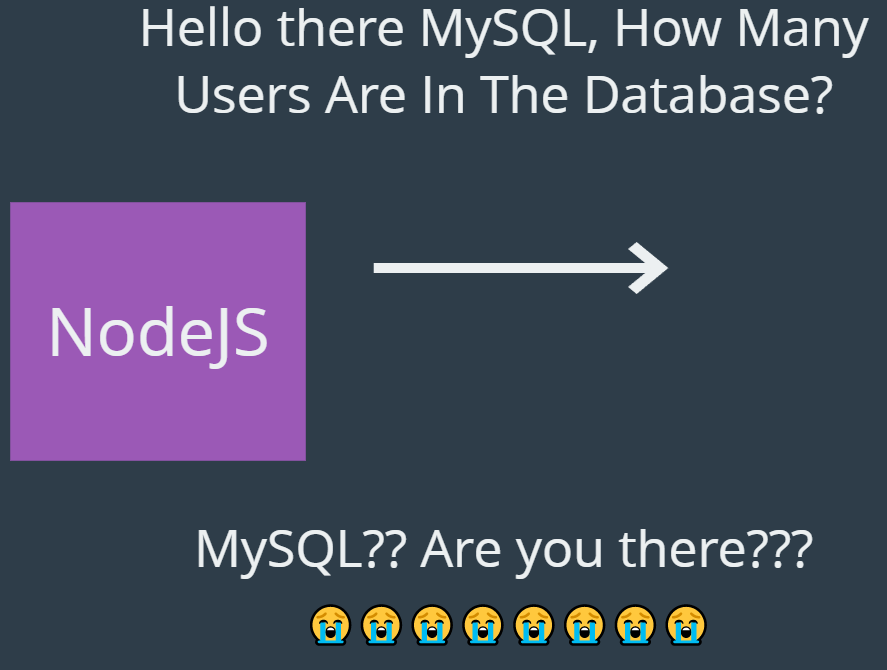
* Faker on its own won’t be able to generate the fake data we need for us. So we know how to generate fake emails and dates
* This pseudocode tells us what we want to go for (not real code!)



* The problem is that the INSERT INTO command only exists in MySQL, and the faker syntax only exists in NodeJS. To solve this, we use the **MySQL node package**
  + The MySQL package (in Node) connects to MySQL and talks to it
  + (Yeah, it’s confusing that the package is called MySQL)
* Here’s are some diagrams of how the package kind of works



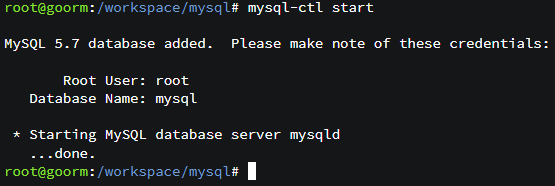




* + The MySQL package within Node will help us handle all of these situations

# Connecting Node to MySQL

* We need to establish communication between the MySQL database and the JavaScript (Node) file
* After connecting, we’ll start by running some simple, silly MySQL queries
* But first, let’s start up MySQL within goorm:
  + Take note of the root user



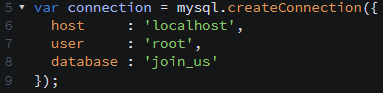
* Now we need to connect to the cli as normal and create a database called **join\_us**
* Now, *how do we get the JavaScript file to talk to the MySQL database that we created*?
  + First, we have to install the MySQL node package from the terminal



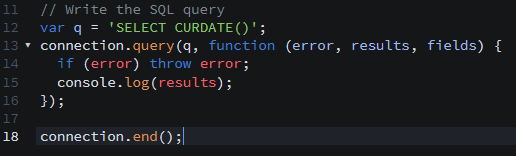
* + Next, we have to require the MySQL package within our JavaScript file



* + Then we have to establish a connection using **createConnection**, which is a method that comes with the mysql package. You will need the user name and the database name

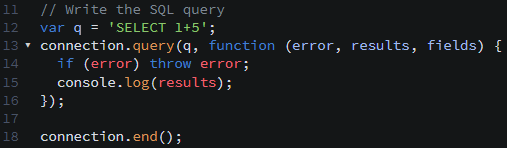


* The file won’t do anything just yet, which is good. But now we’re ready to run queries. Here’s how we write it out:
  + The basic idea is that we have our simply query of “SELECT CURDATE()”, and the rest of the stuff is JavaScript wrapper that is required for that query to function.
  + The **connection.end()** function closes the connection to the database



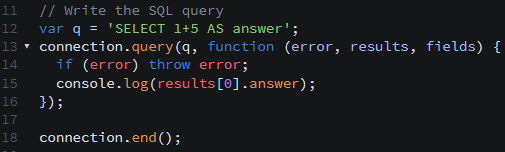


* We now have the tools to talk to our database and do more complex things! Let’s try another trivial example:



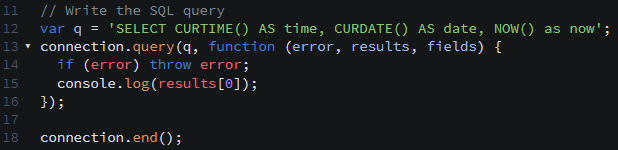


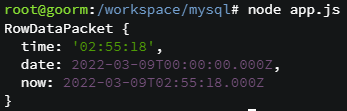
* + Looks good! But note that the return syntax is kind of wonky. What if we just want the straight up answer of “6”? One way we can do it is to alias the answer in our query. The alias will be assigned as an attribute to the result, allowing us to directly access it using attribute notation!
    - In this case, we’ll store that result of the selection as the alias “answer”, and we’ll access it using **.answer**.
    - Note that we also need to use array accession ([0]) to get to the result itself, as opposed to the result data structure. The result is the first item within the result data structure array



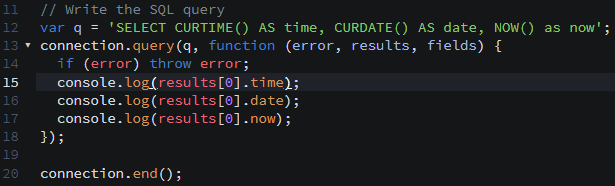


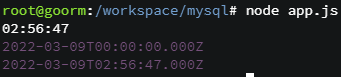
* Let’s do a final example where we “SELECT” multiple items





* + Cool, but what if we want to print them all separately? For that we’ll need to access them individually using array access notation and alias keywords as follows:

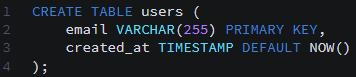


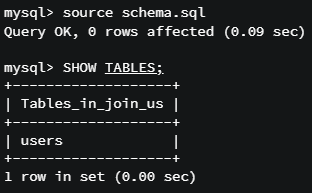


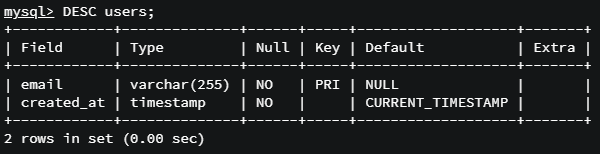
* How did Colt know what code to write to get this connectivity? It’s from the documentation written by the authors of the library
  + This is not MySQL-sanctioned code. It was written by developers and we need to follow the rules created by those developers
  + The documentation can be found here: <https://github.com/mysqljs/mysql>

# Creating Our Users Table in MySQL

* Now we’re ready to create define our schema and create our table in MySQL. After that, we’ll attempt to query it in JavaScript
* Let’s create our table for *users*, where we’ll have a field for **email** and for **created\_at**. We’ll work in a new SQL file called **schema.sql**







* Now we have a *users* table – nice!
* Code summary

CREATE TABLE users (

email VARCHAR(255) PRIMARY KEY,

created\_at TIMESTAMP DEFAULT NOW()

);