

# Introduction to Sequelize and ORM in Node.js

Managing database drivers, managing schematics, maintaining a business' workflow, and validating data can be daunting for any programmer. Along with perpetually changing business requirements, organizing the business logic into database models can be cumbersome. This usually entails the programmer finding all applicable references and updating queries manually. This could be an expensive operation for both the project and the programmer; without proper testing, the modifications could result in errors within the application or erroneous logic, leaving the programmer, the business, and the customer in a state of confusion.

This book will help guide you through the process of installing, building, maintaining, upgrading, extending, querying, and applying database schematics using an **object-relational mapping (ORM)** framework in a Node.js application using the Node.js runtime environment. The book can be read from start to finish in a sequential manner, or if you are more experienced, you can read the chapters that interest you directly. Each chapter complements the previous chapter since we will be creating an entire application from scratch. However, more experienced programmers can skip between chapters with the understanding that there may be “gaps” within their data model and what is shown within the chapter. The concepts and methodologies taught in each chapter, regardless of your data's structure, will still be applicable.

The goal of this chapter is to help you become familiar with what Sequelize is and which capabilities are offered to you from using Sequelize. We will go over the necessary prerequisite steps for installing

applicable libraries, frameworks, runtime engines, and **database management systems (DBMS)**. By the end of this chapter, you will have acquired the knowledge and skillset of installing, configuring, and running an application, under the Node.js runtime with Sequelize, from scratch.

The first chapter of this book will cover the following topics:

- Introducing Sequelize
- Advantages of using Sequelize over other alternatives
- Installing the necessary applications, frameworks, and tools to help get you started
- Configuring Sequelize within an Express application

## Technical requirements

Before we embark on our journey of developing an application with Sequelize, there are a few prerequisites. We will need to install the following:

- A DBMS such as MySQL
- The Node.js runtime library
- A few Node.js packages: Sequelize, Express, and a MySQL driver

## Introducing Sequelize

**Sequelize** (also known as **SequelizeJS**) is an ORM framework that helps connect and correspond your Node.js application to a database. Sequelize has been in development since 2010 by Sascha Depold and is used extensively within *Fortune 100* companies. Throughout the years, the framework has grown to nearly 25,000 *stargazers* on GitHub, with over 900 contributors, and is used by over 300,000 open sourced projects.

Sequelize has been *battle-tested* for performance and security for over a decade and has performed without issues for major retail stores and web

agencies (such as Walmart and Bitnami) even during their highest traffic times of the year.

What started out as a master's thesis turned into a major integral building block of Node.js' ecosystem.

## NOTE

An ORM is a methodology of associating database structures and information using **object-oriented (OO)** decorations and patterns. An ORM's purpose is to help alleviate the differences between DBMSs and to offer some form of abstraction for querying and manipulating data more ergonomically. Typically, an ORM will also come with helper functions to help manage the state of connections, pre-validation of data, and workflows.

The framework follows a **promise-based** approach, which allows programmers to invoke data asynchronously. The promise-based approach offers a more convenient way of managing returned values, or errors, within your application without waiting for the result(s) to return immediately. To learn more about promises and how to program with them, refer to the following link: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise).

## WHAT IS ASYNCHRONOUS?

Think of asynchronous as a way to perform tasks without having to wait for a response before continuing on with another task. When you text message someone, you do not have to wait for their response in order to continue with your day. After you send a message, you usually would not warrant any attention to the correspondence until you receive a signal that there was a response or that the message had failed to send.

Currently, Sequelize supports the following DBMSs: MySQL, MariaDB, Postgres, **Microsoft SQL Server (MSSQL)**, Snowflake, **Database 2 (DB2)**,

and SQLite. An ORM offers more than just a connector to your database. ORMs often offer features such as the following:

- Tooling for migrating schemas and data
- Adapter/plugin support
- Connection pooling
- Eager loading of data
- Managed transactions

Now that we understand *what* Sequelize is and its basic capabilities, we will go over *why* we should use an ORM such as Sequelize over alternative methods such as **data access objects (DAOs)** or querying the database directly. Some of the advantageous capabilities include being able to handle and organize queries within transactions or migrating schematic changes to a database.

## Advantages of using Sequelize over other alternatives

There are many alternative ways of querying the database from your application. There are ORMs, DAOs, raw database drivers, and so on. Each methodology has its pros and cons and caters to different programming styles and conventions. Typically, those who favor *convention over configuration* tend to gravitate toward ORMs, while those who favor configuration tend to use DAO frameworks or raw database drivers.

An ORM can handle data validation, similar to DAOs, with additional features such as reading and writing from a database using a driver. With ORMs, you would not need to manage query statements manually, which could save you time over the DAO or raw connection methods.

NOTE

An ORM is not mutually exclusive to DAOs. You can think of DAOs as being explicit as opposed to being implicit and presumptuous. A DAO only provides an *interface* for your data. It does not involve how/where you read or write the data (the database driver), nor will it concern itself with the data's integrity unless the application manually invokes some form of data validation outside of the DAO's scope.

When using an ORM such as Sequelize, you will have the following features without any additional code:

- Transaction handling
- Connection pooling
- Model/data validation
- Data integrity (outside of DBMS' scope of **foreign keys (FKs)**, unique constraints, and so on)
- Eager loading
- Schematic migration and cascading
- Optimistic locking

Using a DAO or a raw database driver will forfeit these features, and you will have to build these solutions yourself. Using an ORM such as Sequelize will help you build your project with more efficiency and efficacy.

So far, we have covered the *what* and *why* for Sequelize; now, we will be going over the *how* for installing the necessary prerequisites for our application.

## Installing the necessary applications, frameworks, and tools to help get you started

Our application will require customers to view information from a centralized source, and we will need to capture information that they have entered into our database. Usually, customers can either view your product/services via an application that they install on their machine or they can use a browser to visit our website. Node.js is a good choice for building web applications, which is what we'll be building throughout this book, due to its **central processing unit (CPU)**-bound limitations and ease of context switching between *frontend development* (what is displayed to the end user) and *backend development* (what the end user does not see but still invokes) owing to Node.js being JavaScript. We will need to install the following applications/programs in order to get started:

- A DBMS (we will be installing MySQL)
- Node.js runtime
- Sequelize and Express

## Installing MySQL

This next section will go over the installation process for MySQL on three different operating system distributions: Microsoft Windows, macOS, and Linux. MySQL was chosen due to the ease of installation (no need to mess with configurations or **access-control lists (ACLs)**). Do not let those points discourage you from using a different database. For the most part, Sequelize will be able to gracefully translate from one DBMS to another, and the majority of this book will use common/standard **Structured Query Language (SQL)** methods.

### Windows

The MySQL installer for Microsoft Windows can be found here:

<https://dev.mysql.com/downloads/mysql/5.7.html>

NOTE

The default **Uniform Resource Locator (URL)** for downloading Windows' MySQL installer is currently at version **8.0.26**. This book uses version **5.7**, but other versions of MySQL should still work appropriately as long as the Node.js MySQL driver is compatible with that version.

Once we are finished downloading and opening the installer application, you will be greeted with the **Choosing a Setup Type** screen. We will want to select the **Developer Default** and **Install all products** options, as illustrated in the following screenshot:

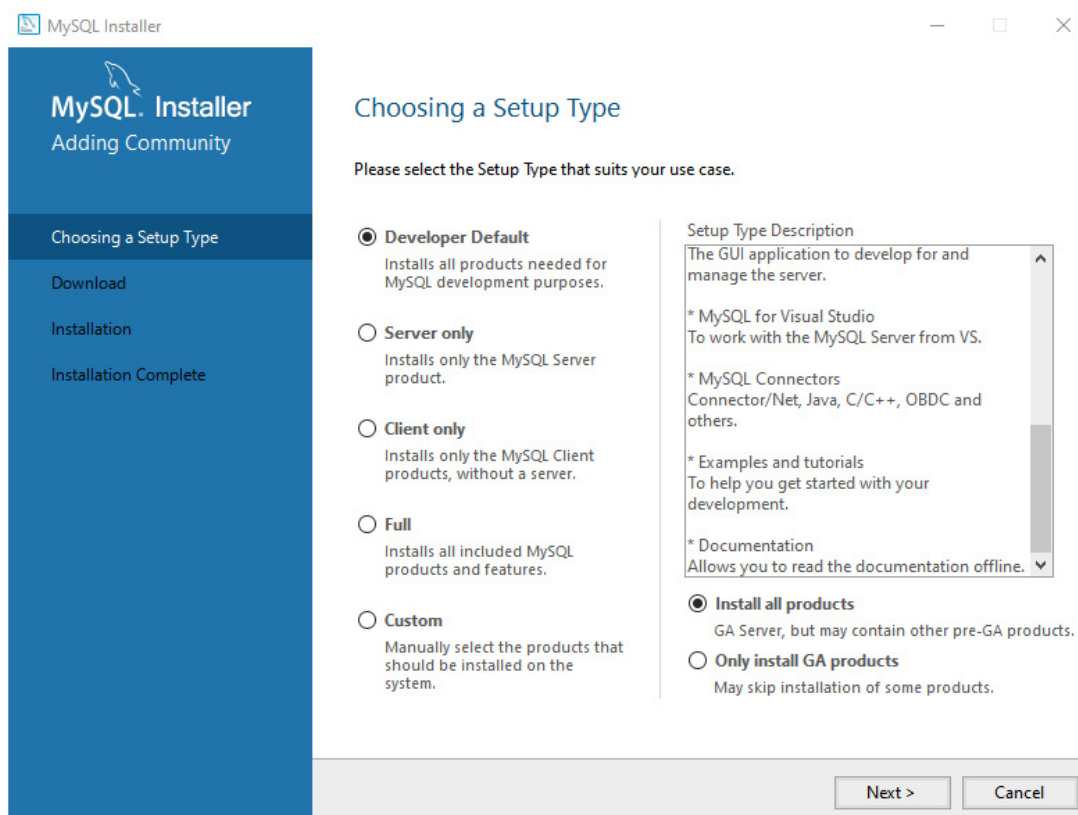


Figure 1.1 – Windows MySQL Installer: Choosing a Setup Type

If you have Python or Visual Studio installed on your computer, you may be greeted with a **Check Requirements** step (see *Figure 1.2*). If you are using Visual Studio as your **integrated development environment (IDE)**, then you may install the necessary products, but it is not a requirement. Throughout your projects, you may come across a utility that is written in Python that interacts with your database (for example, most data science-related libraries/frameworks). By selecting the

**Connector/Python** option shown in the following screenshot, we can avoid potential headaches in the future:

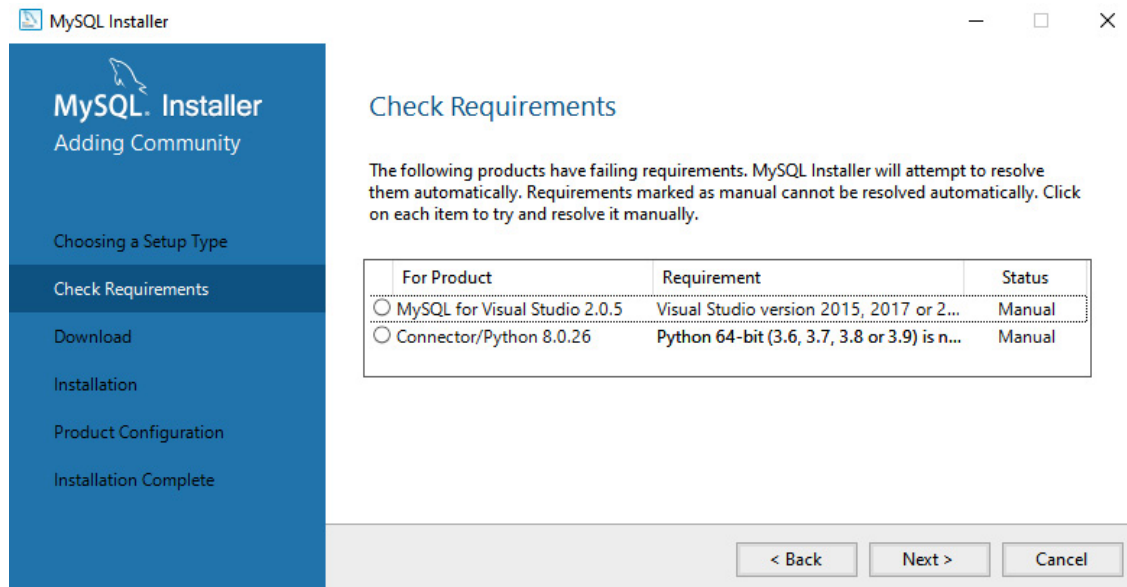


Figure 1.2 – Windows MySQL Installer: Check Requirements

The next section should be the **Download** step. The main products that we will be required for the contents of this book are listed here:

- **MySQL Server**
- **MySQL Workbench** (for a **graphical user interface (GUI)** to our database)
- **MySQL Shell**

You can see the aforementioned products in the following screenshot:



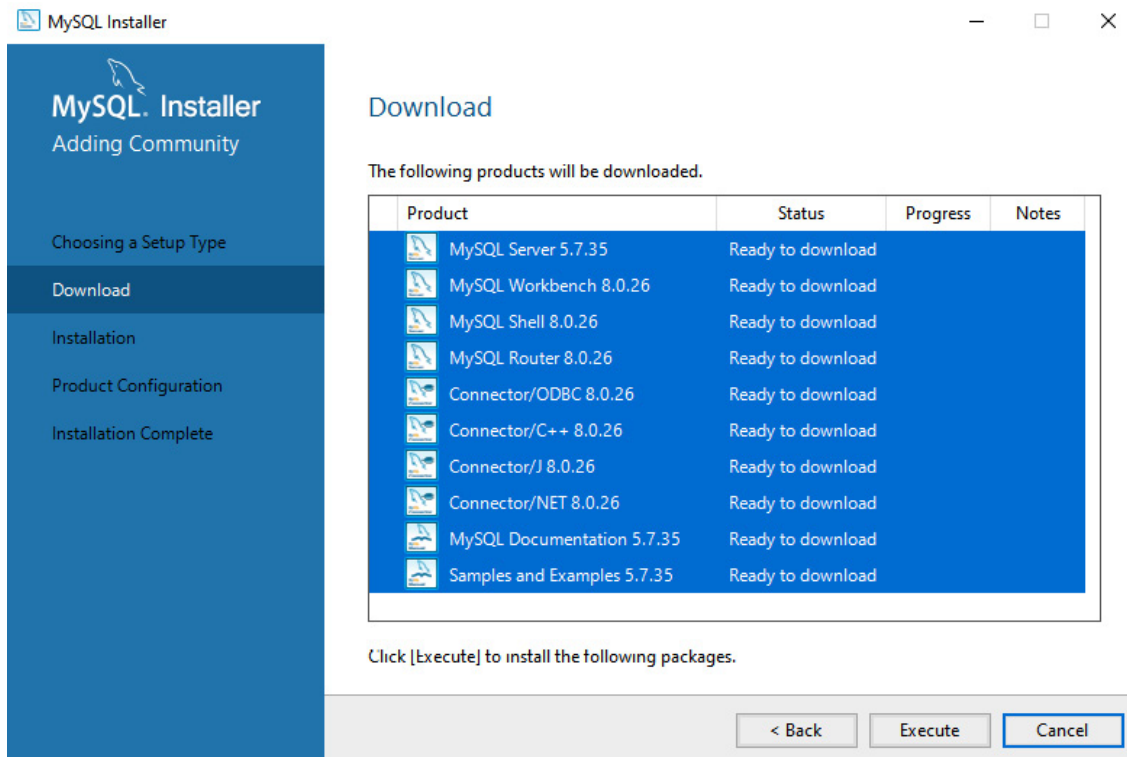


Figure 1.3 – Windows MySQL Installer: Download

#### NOTE

If you are new to MySQL, it may be a good idea to download the **MySQL Documentation** and **Samples and Examples** packages.

After we have finished downloading our packages, we will be entering our configuration details for each applicable selected product (for example, **MySQL Server** and **Samples and Examples**). For the majority of the configuration settings, we will be using the default values; however, there will be some steps that will require your intervention. You can see an overview of this in the following screenshot:

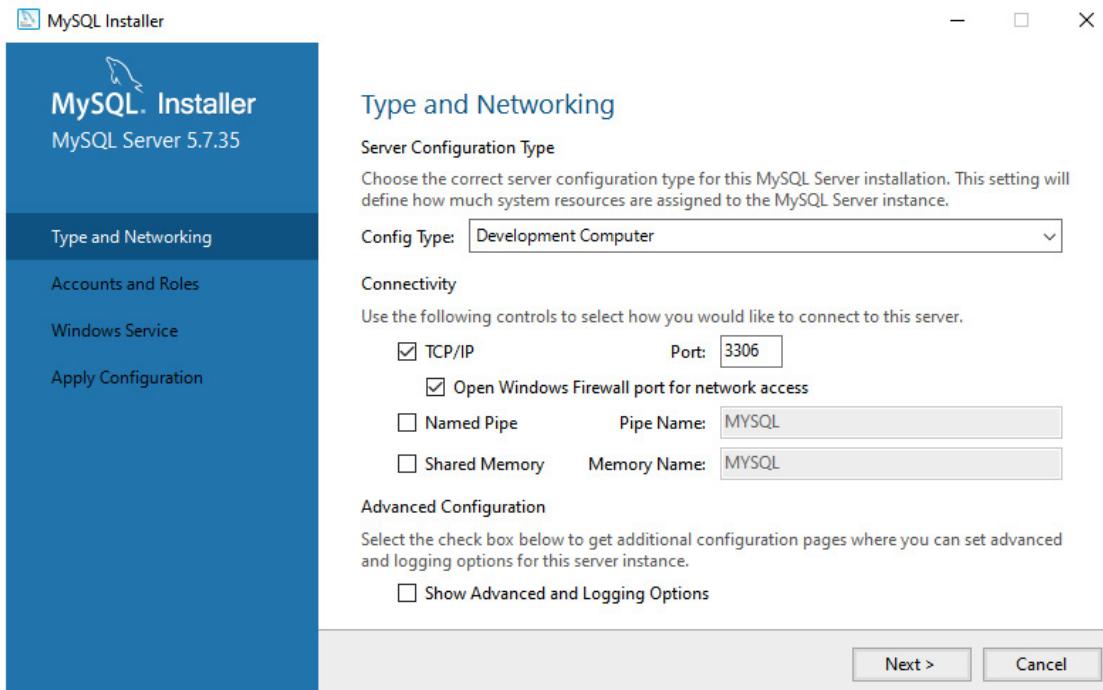


Figure 1.4 – Windows MySQL Installer: Type and Networking

From the **MySQL Server** configuration wizard, we will want the following settings (as shown in *Figure 1.4*):

- **Config Type: Development Computer**
- **TCP/IP:** Checked
- **Port: 3306**
- **Open Windows Firewall port for network access:** Optional

The next part of the MySQL Server configuration step is to declare your MySQL root password and user accounts. Make sure to keep this information in a safe place in case you run into administration issues throughout your projects. If you forget the MySQL root password, there are several methods for resetting the password, as explained here:

<https://dev.mysql.com/doc/mysql-windows-excerpt/5.7/en/resetting-permissions-windows.html>.

For setting up a MySQL user account with a role, you will be greeted with the following **Accounts and Roles** screen:

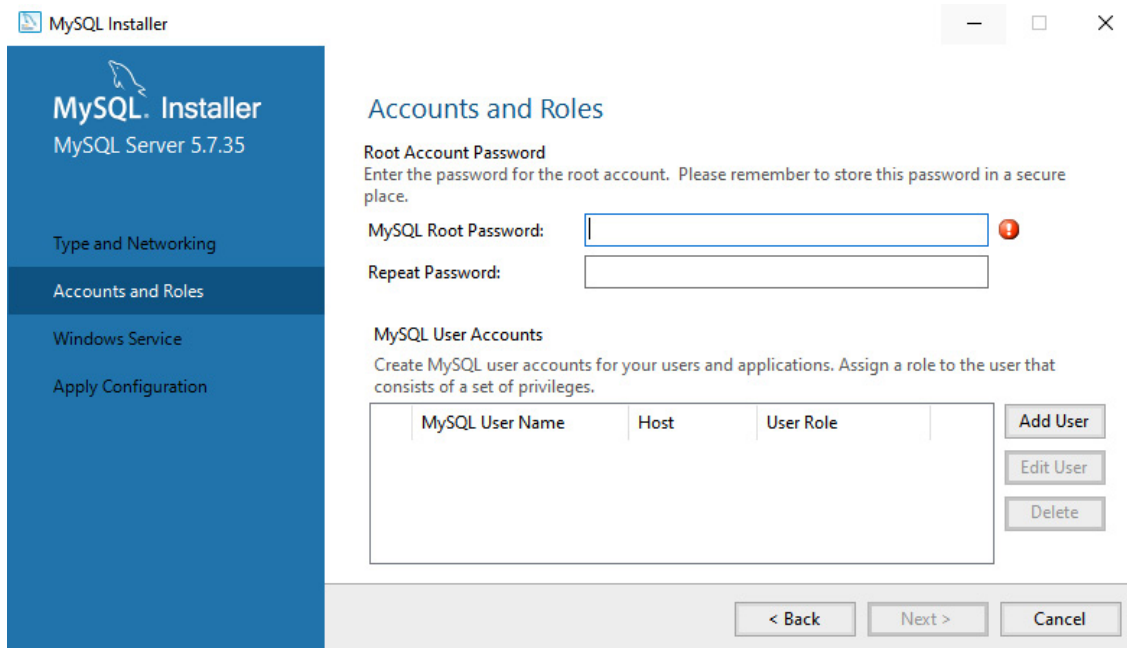


Figure 1.5 – Windows MySQL Installer: Accounts and Roles

Within the **MySQL User Accounts** section, you will need to click on the **Add User** button (near the right side of the window, as shown in *Figure 1.5*) and type in a username and password that you will memorize for when we initialize our Node.js application. When you are finished adding the appropriate root password and MySQL user account(s), we can proceed to the next step.

Next, the installation process will offer a **Configure MySQL Server as a Windows Service** option, as illustrated in the following screenshot. **Windows Service** is a **process control system (PCS)** that will also orchestrate background processes (in the Unix/Linux world, these are referred to as *daemons*):

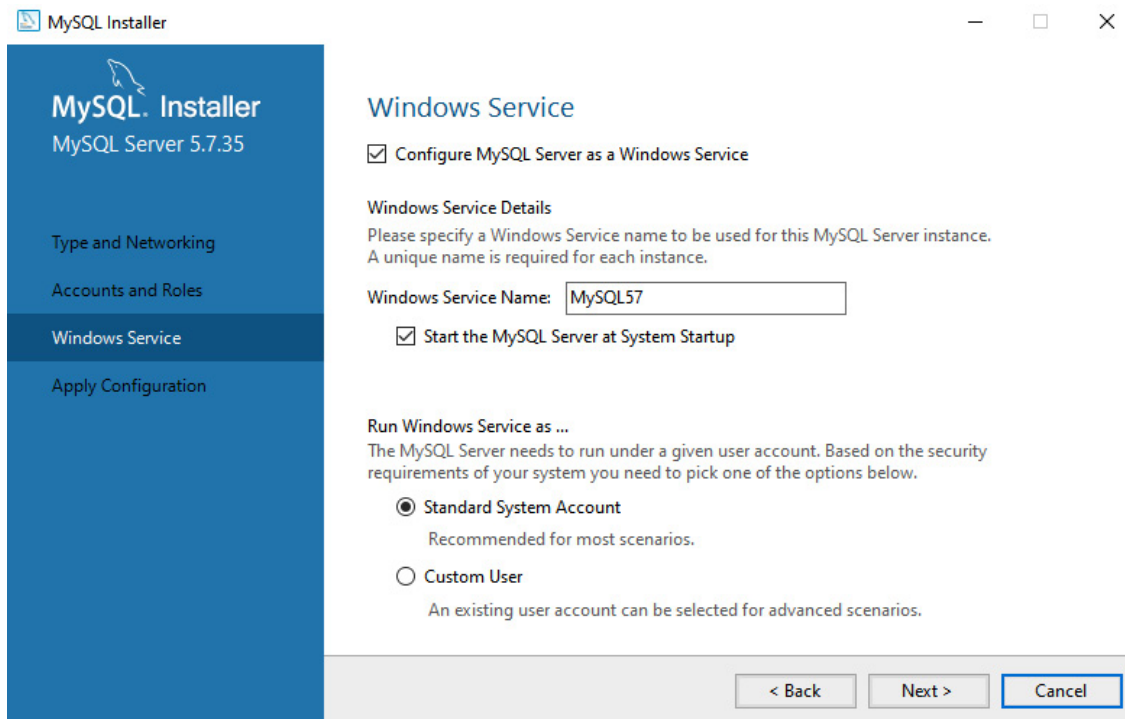


Figure 1.6 – Windows MySQL Installer: Windows Service

We will want to ensure the following parameters are configured (as shown in *Figure 1.6*):

- **Configure MySQL Server as a Windows Service:** Checked
- **Start the MySQL Server at System Startup:** Checked
- **Standard System Account** selected under the **Run Windows Service as...** section

Click on **Next >** to apply our configurations for the MySQL server. If you selected additional packages to install earlier, you may be prompted with additional screens asking for more configuration settings and parameters.

#### NOTE

If you selected the **MySQL Router** package from the previous section, the installation process will ask you for information on how you would like to set up a cluster environment. It is not recommended to install this package unless you are a database administrator or you are setting up a production environment. Simply

uncheck the **Bootstrap MySQL Router for use with InnoDB cluster** option and click **Finish** to proceed without installing MySQL under a cluster environment.

If the **Samples and Examples** package was selected for installation, we will be prompted with a screen that will allow us to enter our MySQL username and password. You may use your *root credentials* for the username and password input fields and click on the **Next >** button to continue. An overview of the screen is provided in the following screenshot:

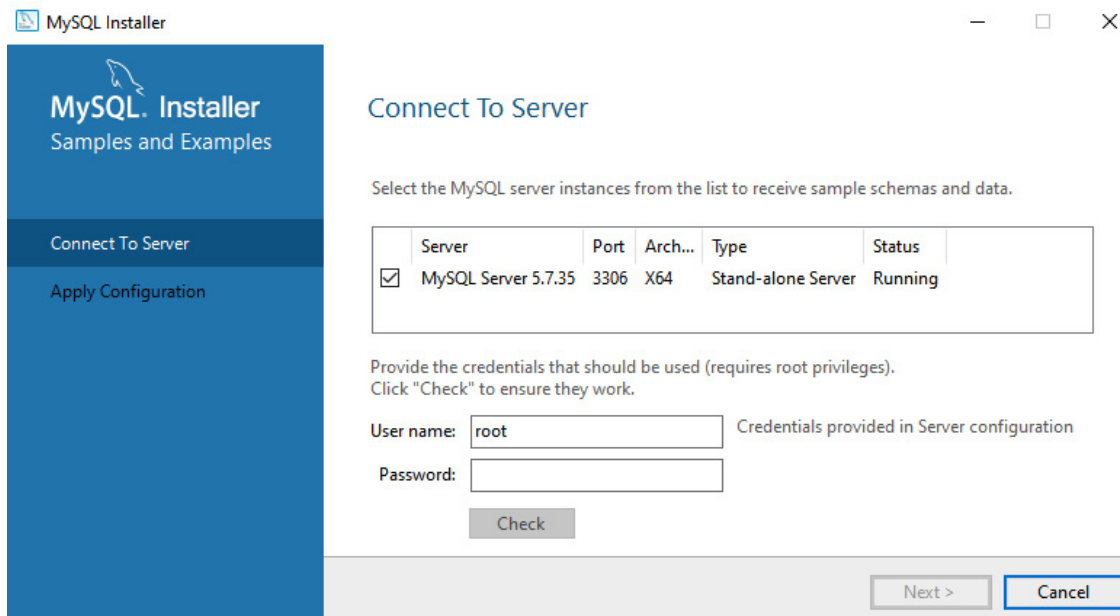


Figure 1.7 – Windows MySQL Installer: Connect To Server

## macOS

There are a couple of ways to install MySQL on a macOS machine. The first way is to download and install MySQL from a **Disk iMaGe (DMG)** file, while another method is by using a package manager such as Homebrew. We will explore both options.

### Installing from disk image

You can find the appropriate disk image from the following URL: <https://dev.mysql.com/downloads/mysql/> (x86 for Intel CPUs and **Advanced RISC Machine (ARM)** for M1 CPUs).

## NOTE

If you cannot find version 5.7 for MySQL, you will find the appropriate DMG file from MySQL's archive link:

<https://downloads.mysql.com/archives/community>.

However, the macOS installation packages may not be available to download for the most recent 5.7 versions. At the time of writing this book, versions 5.7.34, 5.7.33, and 5.7.32 are not available as a DMG package (5.7.31 is available to download). Any applicable 5.7 version should be compatible with this book's instructions and installation procedures.

If you are asked about installing a preference panel throughout the installation process, we recommend you do so. Otherwise, we will need to consult the **Installing a MySQL Launch Daemon** page, located at

<https://dev.mysql.com/doc/refman/5.7/en/macos-installation-launchd.html>.

After downloading and opening the DMG file, we will want to open the **package (pkg)** file, which will start our installation process. Depending on your macOS version, you may be prompted with a “[package name]” **can't be opened because Apple cannot check it for malicious software** screen, as shown here:



Figure 1.8 – Apple cannot identify the package for maliciousness

If this is the case for you, go to **Apple | Security & Privacy**, and the window should have an **Open Anyway** button next to “mysql....pkg” was blocked from use because it is not from an identified developer., as shown in the following screenshot:

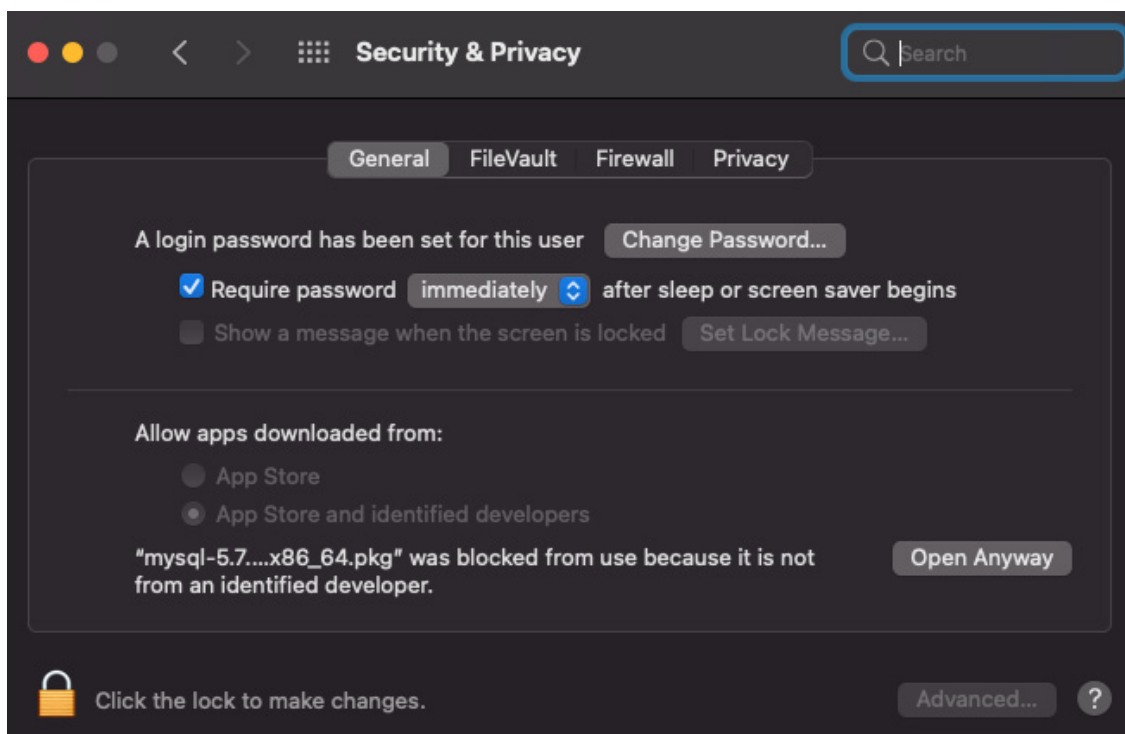


Figure 1.9 – Bypassing unidentified package installations

Once the installation package opens again, you may be prompted with another alert from Apple. Click on **Open** to continue with the installation process. After continuing and reading the **software license agreement (SLA)**, you may select the default installation location. Clicking on **Install** may prompt for your administrative password, as illustrated in the following screenshot:

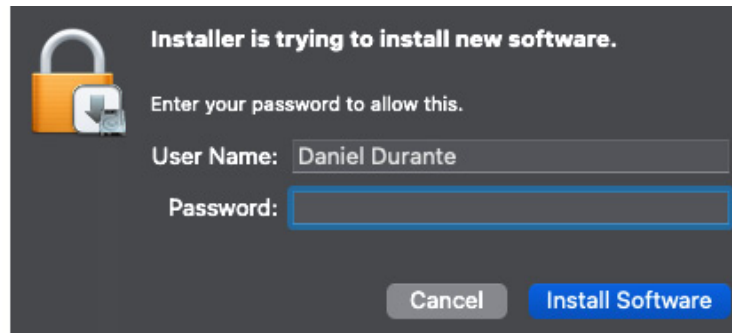


Figure 1.10 – MySQL installation asking for administrative permission

Once the MySQL installer finishes, an alert dialog will appear with a temporary password. An example is shown in the following screenshot. Make sure to take note of the temporary password for when we log in to the MySQL server:

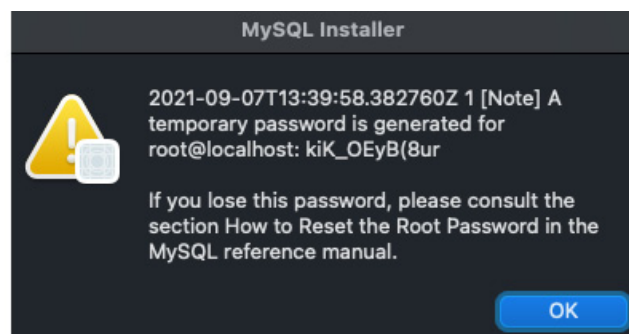


Figure 1.11 – MySQL installation providing a temporary root password

## Installing from Homebrew

Using Homebrew over traditional package installers can help keep your packages up to date without manual intervention, along with validating package installations and binaries. To install MySQL through Homebrew,



we will need to install Homebrew on our local machine. Within the terminal (located in **Applications > Utilities**), simply type in the following:

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

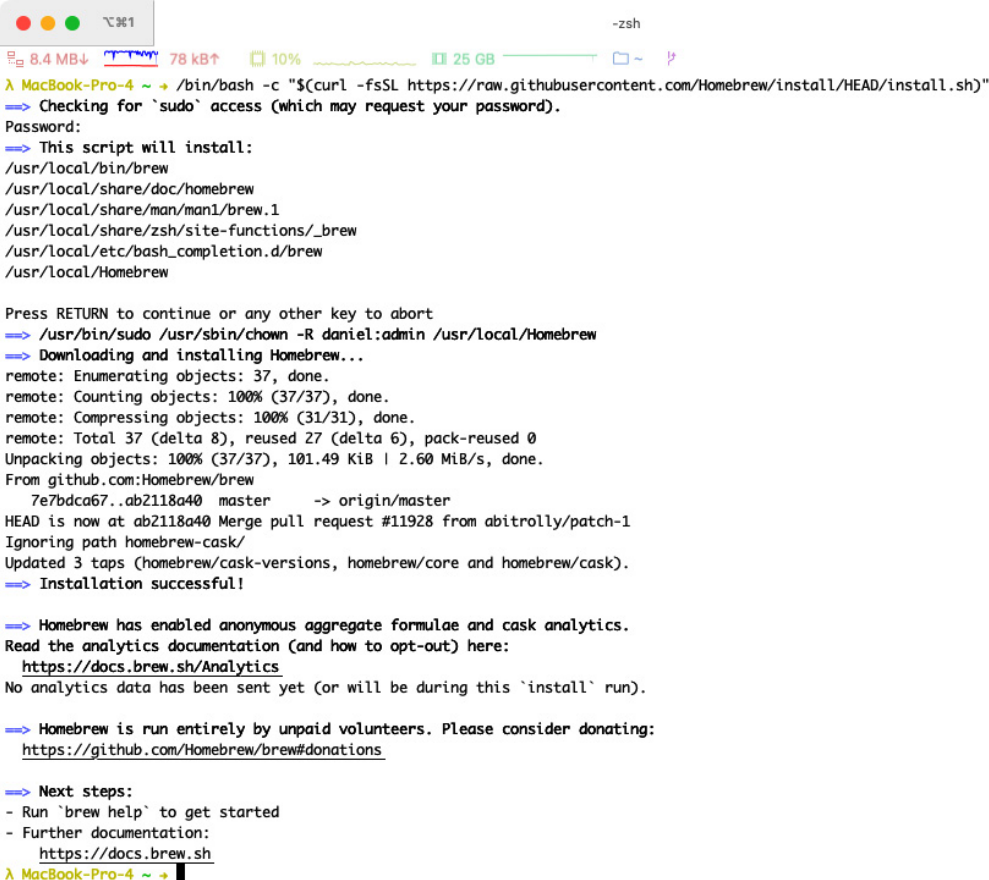
## NOTE

It is always a good idea to double-check an external script's contents before running commands from it. A web page can redirect to anywhere, including malicious scripts that could lead to data breaches or something more nefarious.

When installing Homebrew, you may come across the following message:

```
==> Checking for `sudo` access (which may request your password)
```

You can see an illustration of this in the following screenshot:



```
MacBook-Pro-4 ~ % /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
=> Checking for `sudo` access (which may request your password).
Password:
=> This script will install:
/usr/local/bin/brew
/usr/local/share/doc/homebrew
/usr/local/share/man/man1/brew.1
/usr/local/share/zsh/site-functions/_brew
/usr/local/etc/bash_completion.d/brew
/usr/local/Homebrew

Press RETURN to continue or any other key to abort
=> /usr/bin/sudo /usr/sbin/chown -R daniel:admin /usr/local/Homebrew
=> Downloading and installing Homebrew...
remote: Enumerating objects: 37, done.
remote: Counting objects: 100% (37/37), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 37 (delta 8), reused 27 (delta 6), pack-reused 0
Unpacking objects: 100% (37/37), 101.49 KiB | 2.60 MiB/s, done.
From github.com:Homebrew/brew
  7e7bdca67..ab2118a40 master -> origin/master
HEAD is now at ab2118a40 Merge pull request #11928 from abitrolly/patch-1
Ignoring path homebrew-cask/
Updated 3 taps (homebrew/cask-versions, homebrew/core and homebrew/cask).
=> Installation successful!

=> Homebrew has enabled anonymous aggregate formulae and cask analytics.
Read the analytics documentation (and how to opt-out) here:
  https://docs.brew.sh/Analytics
No analytics data has been sent yet (or will be during this `install` run).

=> Homebrew is run entirely by unpaid volunteers. Please consider donating:
  https://github.com/Homebrew/brew#donations

=> Next steps:
- Run `brew help` to get started
- Further documentation:
  https://docs.brew.sh
MacBook-Pro-4 ~ %
```

Figure 1.12 – Installing Homebrew on macOS

You can either enter in your password here or before installing Homebrew, run **sudo <anything>** (for example, **sudo ls**), enter in your password, and then run the installation command. The user must have administrator access before continuing.

For this book, we will install MySQL version 5.7. Other versions of MySQL should be compatible with the book's code base, as previously noted. To install version 5.7 explicitly, run the following command:

```
brew install mysql@5.7
```

There may be additional steps and commands to run in order to set up your instance properly, as illustrated in the following screenshot. The book's contents will not require library/header files for compilation, nor for **pkg-config** to be configured. As a general rule, it is recommended to run **mysql\_secure\_installation** and go through the prompts for adding a root password, but it is not a requirement:



```
λ MacBook-Pro-4 ~ → brew install mysql@5.7
➡ Downloading https://ghcr.io/v2/homebrew/core/mysql/5.7/manifests/5.7.35-1
Already downloaded: /Users/daniel/Library/Caches/Homebrew/downloads/8d6a41ecd64a64e4997897ce7c853b2f9c92646544b8f2df7260d72506eff4d9
--mysql@5.7-5.7.35-1.bottle.manifest.json
➡ Downloading https://ghcr.io/v2/homebrew/core/mysql/5.7/blobs/sha256:983b9015ebe2bf32c43eb309de1712a289eaf22c6a17de3e5f34c8d0f918
Already downloaded: /Users/daniel/Library/Caches/Homebrew/downloads/d8237be5e54bf7044ca0e87a8e7d791325178341af709efbda5f54170b2efb14
--mysql@5.7--5.7.35.big_sur.bottle.1.tar.gz
➡ Pouring mysql@5.7--5.7.35.big_sur.bottle.1.tar.gz
➡ Caveats
We've installed your MySQL database without a root password. To secure it run:
  mysql_secure_installation

MySQL is configured to only allow connections from localhost by default

To connect run:
  mysql -uroot

mysql@5.7 is keg-only, which means it was not symlinked into /usr/local,
because this is an alternate version of another formula.

If you need to have mysql@5.7 first in your PATH, run:
  echo 'export PATH="/usr/local/opt/mysql@5.7/bin:$PATH"' >> ~/.zshrc

For compilers to find mysql@5.7 you may need to set:
  export LDFLAGS="-L/usr/local/opt/mysql@5.7/lib"
  export CPPFLAGS="-I/usr/local/opt/mysql@5.7/include"

For pkg-config to find mysql@5.7 you may need to set:
  export PKG_CONFIG_PATH="/usr/local/opt/mysql@5.7/lib/pkgconfig"

To restart mysql@5.7 after an upgrade:
  brew services restart mysql@5.7
Or, if you don't want/need a background service you can just run:
  /usr/local/opt/mysql@5.7/bin/mysqld_safe --datadir=/usr/local/var/mysql
➡ Summary
📦 /usr/local/Cellar/mysql@5.7/5.7.35: 320 files, 234.6MB
λ MacBook-Pro-4 ~ →
```

Figure 1.13 – Installing MySQL with Homebrew on macOS

Next, we will need a way to manage our MySQL service. There are two options available to us, as outlined here:

- Manually create launch daemon configuration files. More information on how this can be achieved is available here:  
<https://dev.mysql.com/doc/refman/5.7/en/macos-installation-launchd.html>.
- We can use a Homebrew extension known as **services** to manage launch configurations automatically by executing the following command:

```
brew tap homebrew/services
```

In order to start the MySQL service, we need to run the following command:

```
brew services start mysql@5.7
```

If you prefer a GUI version of managing your services, there is an application called **brew-services-menubar** that can be installed via Homebrew's Cask extension, as shown in the following code snippet:

```
brew install --cask brewservicesmenubar
```

NOTE

If you prefer to use a GUI when interfacing/querying databases, there is a free application called Sequel Pro that is available for downloading here:

<http://www.sequelpro.com/>

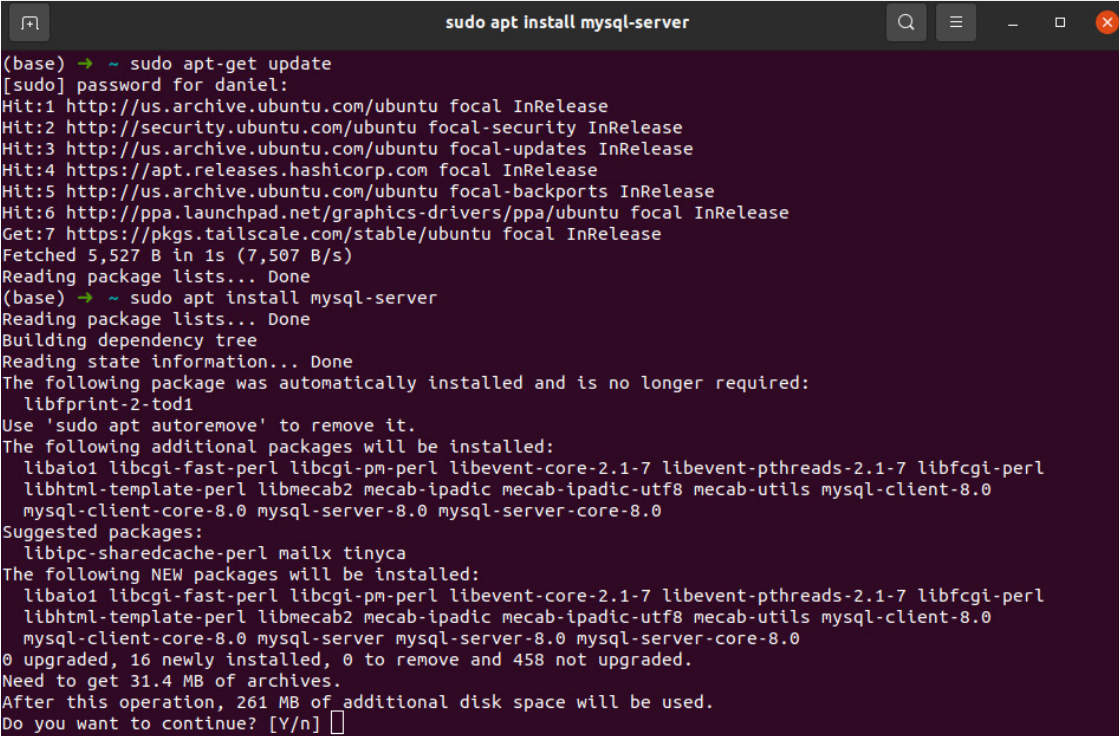
## Linux

There are numerous distributions of Linux; for this book, we will be using Ubuntu (any Debian distribution should be applicable with the same commands). If you are using a different distribution, please refer to this page for instructions on how to install MySQL for your operating system:

<https://dev.mysql.com/doc/refman/5.7/en/linux-installation.html>.

Within the terminal, run the following commands (these are also shown in the screenshot that follows):

```
sudo apt-get update
sudo apt install mysql-server
```



```
(base) → ~ sudo apt-get update
[sudo] password for daniel:
Hit:1 http://us.archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:4 https://apt.releases.hashicorp.com focal InRelease
Hit:5 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease
Hit:6 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu focal InRelease
Get:7 https://pkgs.tailscale.com/stable/ubuntu focal InRelease
Fetched 5,527 B in 1s (7,507 B/s)
Reading package lists... Done
(base) → ~ sudo apt install mysql-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libfprint-2-tod1
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libaio1 libcgi-fast-perl libcgi-pm-perl libevent-core-2.1-7 libevent-pthreads-2.1-7 libfcgi-perl
  libhtml-template-perl libmecab2 mecab-ipadic mecab-ipadic-utf8 mecab-utils mysql-client-8.0
  mysql-client-core-8.0 mysql-server-8.0 mysql-server-core-8.0
Suggested packages:
  libipc-sharedcache-perl mailx tinycd
The following NEW packages will be installed:
  libaio1 libcgi-fast-perl libcgi-pm-perl libevent-core-2.1-7 libevent-pthreads-2.1-7 libfcgi-perl
  libhtml-template-perl libmecab2 mecab-ipadic mecab-ipadic-utf8 mecab-utils mysql-client-8.0
  mysql-client-core-8.0 mysql-server mysql-server-8.0 mysql-server-core-8.0
0 upgraded, 16 newly installed, 0 to remove and 458 not upgraded.
Need to get 31.4 MB of archives.
After this operation, 261 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Figure 1.14 – Installing MySQL Server on Ubuntu

After MySQL has finished its installation, we will need to initialize a database to store all of our model’s schemas and information. Some ORMs and DBMSs will refer to databases as “schemas” (not to be confused with a model’s schema, which is referred to as “attributes” in Sequelize specifically).

## Creating a database

Now that we have finished installing the MySQL DBMS engine on our local machine, we can start creating a database with some tables. Before creating tables, we will need to go over the various types of MySQL engines. Luckily for us, the following is applicable to all operating systems in the same way.

By default, MySQL will create an InnoDB database type (or, in MySQL terms, engine). Database engines are associated with the database's table on MySQL (and not the entire database itself). This is useful when you know the trade-offs between a read-heavy table with no constraints (for example, news articles) and a write-heavy table (for example, a chat-room). For the sake of brevity, we will go over the main three database engines, as follows:

- **InnoDB:** A database engine with transactional queries and FK support. Transactional queries are useful for executing a query, or several queries, with atomicity. We will go into further details about transactions and FKs in a later chapter.
- **MyISAM:** If the majority of your database's operations are read-related and you do not require any data constraints, this would be a preferred database engine to use.
- **HEAP:** The data stored within these tables is contained within the machine's memory. This database engine is useful if you had to query against temporary data quickly. MySQL will not manage memory allocations for you, so it is important to remember to delete tables when they are no longer in use (and that the data fits into the machine's available memory).

## NOTE

You can always check your local MySQL server's default engine type by entering the following command within a MySQL client:

```
SELECT @@default_storage_engine;.
```

You may skip this section and use the Sequelize **command-line interface's (CLI's)** (installation instructions are given later within this chapter) **db:create** command, as long as the applicable MySQL user has the appropriate permissions. For the intent of becoming familiar with the terminal, we will create the database using command lines, as shown in the next screenshot.

Log in to the MySQL server with the following command (you may be prompted to enter in a password, or the additional **-p** parameter is required to enter in a password):

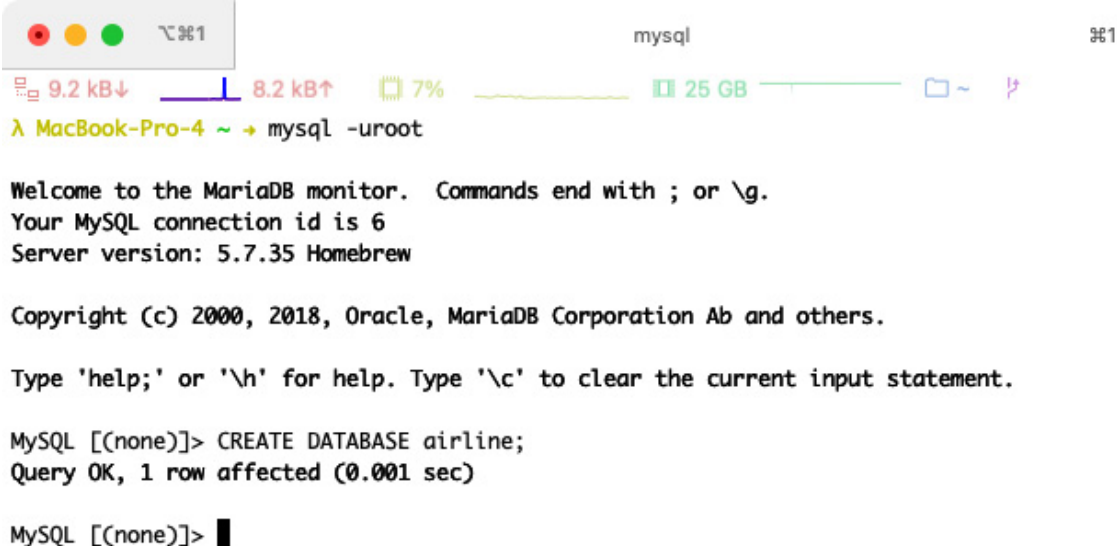
```
mysql --user=root
```

We can create our database by executing the following SQL command within the MySQL client Command Prompt:

```
CREATE DATABASE airline;
```

FOR WINDOWS USERS

Most of these commands are executable via the Command Prompt or PowerShell applications. These applications can be accessed from the **Start** menu (for example, **Start > All Programs > Accessories > Windows PowerShell**).



```
mysql

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.35 Homebrew

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> CREATE DATABASE airline;
Query OK, 1 row affected (0.001 sec)

MySQL [(none)]> 
```

Figure 1.15 – Creating a database

If you are using a Windows machine, you may use any terminal application of your choice (Command Prompt, PowerShell, and so on), or you can use MySQL Workbench, as shown in the following screenshot, which we installed in the previous section:

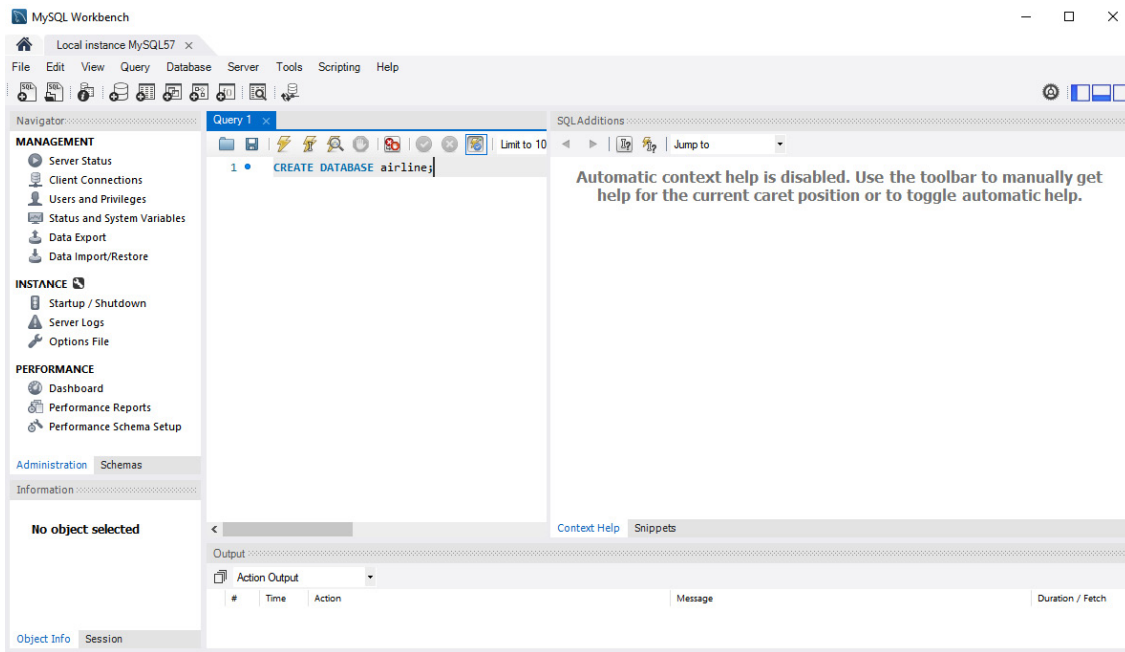


Figure 1.16 – MySQL Workbench: Creating a database

## NOTE

To execute a query using MySQL Workbench, there is a *thunderbolt* icon within the query's toolbar (the icon is usually next to the *save* icon). Your query's results will appear at the bottom of your screen in the **Output** section.

## Installing Node.js

At the time of writing this book, the **long-term support (LTS)** version of Node.js is 16. Throughout this book, we will be using this version of Node.js, but the code base should still execute without issues using other releases. All of the corresponding operating system installations of Node.js can be found here: <https://nodejs.org/en/download/>.

## NOTE

If the LTS version of Node.js is no longer version 16 and you want to use the same version as this book, you can download previous Node.js versions here: <https://nodejs.org/en/download/releases/>.



For managing multiple Node.js versions on one machine, there is an application called **Node Version Manager (NVM)** that can handle and maintain several versions of Node.js on the same machine. For more information, you can visit their repository at <https://github.com/nvm-sh/nvm>.

## Windows

After we are done downloading and opening the Node.js Windows installer, we will be prompted with the following screen:

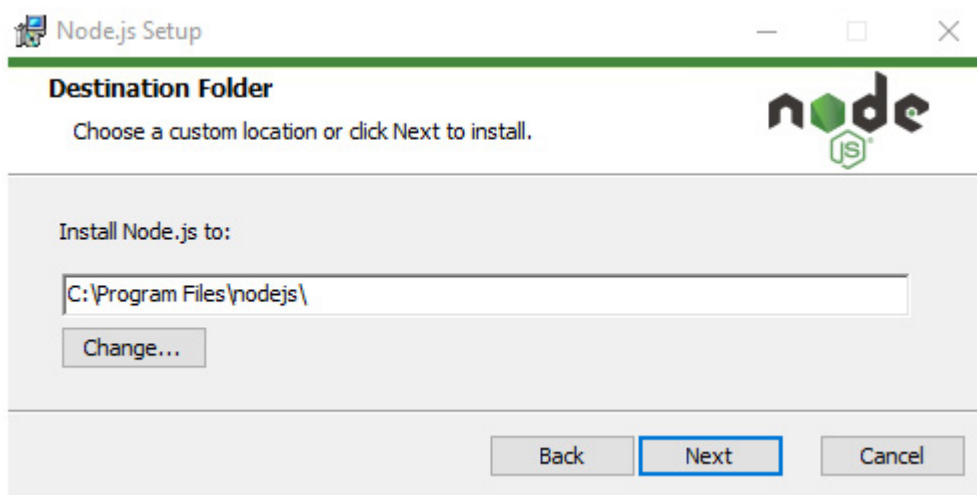


Figure 1.17 – Windows Node.js installer: Destination Folder

Clicking on **Next** will bring us to the **Custom Setup** step of the installation. Ensure that you are installing/configuring the following:

- **Node.js runtime**
- **npm package manager**
- **Add to PATH**

You can see an overview of this screen here:



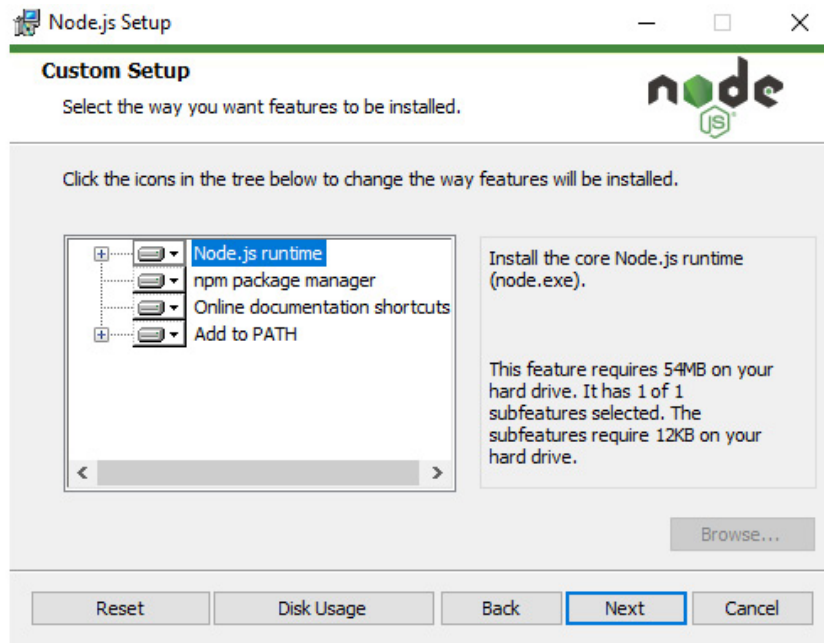


Figure 1.18 – Windows Node.js installer: Custom Setup

After the **Custom Setup** step, we will be brought to a **Tools for Native Modules** section. By default, the checkbox for installing the necessary tools is unchecked. For development purposes, we will want to make sure that the automatic installation option is checked, as depicted in the following screenshot:

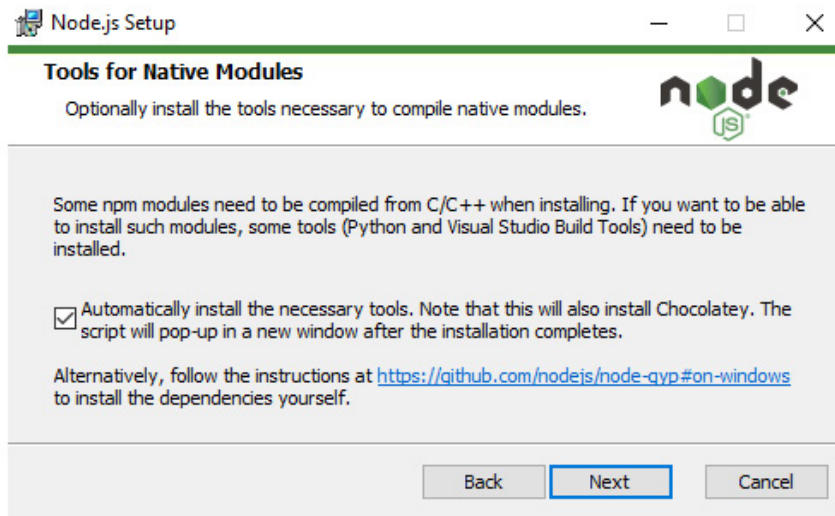


Figure 1.19 – Windows Node.js installer: Tools for Native Modules

Selecting the automatic tool installation will prompt a PowerShell window to appear, as illustrated in the next screenshot, showing you the status of installation progress for Chocolatey, .NET packages, Python dependencies, and so on.

```
Administrator: Windows PowerShell
[6380:0029][2021-09-01T11:29:58] Package Microsoft.VisualStudio.Debugger.DbgHelp.Win8 is not applicable: The current OS
Version '10.0.19043.0' is not in the supported version range '[6.1,6.3]'.
[6380:0029][2021-09-01T11:29:58] Package Microsoft.VisualStudio.Debugger.Remote.DbgHelp.Win8 is not applicable: The curr
ent OS Version '10.0.19043.0' is not in the supported version range '[6.1,6.3]'.
[6380:0029][2021-09-01T11:29:58] Package Microsoft.VisualStudio.Debugger.Remote.DbgHelp.Win8 is not applicable: The curr
ent OS Version '10.0.19043.0' is not in the supported version range '[6.1,6.3]'.
[6380:0029][2021-09-01T11:29:58] Package Microsoft.VisualStudio.NuGet.PowershellBindingRedirect is not applicable: The c
urrent OS Version '10.0.19043.0' is not in the supported version range '[6.1,6.2]'.
[6380:0029][2021-09-01T11:29:58] Shutting down the application with exit code 0
[6380:0001][2021-09-01T11:29:58] Releasing singleton lock.
[6380:0001][2021-09-01T11:29:58] Releasing singleton lock succeed.
[6380:0001][2021-09-01T11:29:58] Releasing singleton lock.
[6380:0001][2021-09-01T11:29:58] Singleton lock does not exist. Releasing singleton lock skipped.
[6380:0001][2021-09-01T11:29:58] Closing the installer with exit code 0
[6380:0001][2021-09-01T11:29:58] Exit code: 0
[6380:0001][2021-09-01T11:29:58] Cleared previous session ID.
visualstudio2017-workload-vctools has been installed.
visualstudio2017-workload-vctools may be able to be automatically uninstalled.
The upgrade of visualstudio2017-workload-vctools was successful.
Software install location not explicitly set, could be in package or
default install location if installer.
Chocolatey upgraded 17/17 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
Upgraded:
- chocolatey-dotnetfx.extension v1.0.1
- kb3033929 v1.0.5
- python3 v3.9.7
- visualstudio2017buildtools v15.9.38.0
- chocolatey-windowsupdate.extension v1.0.4
- vccredist140 v14.29.30133
- kb2999226 v1.0.20181019
- visualstudio-installer v2.0.1
- kb2919355 v1.0.20160915
- chocolatey-core.extension v1.3.5.1
- kb2919442 v1.0.20160915
- visualstudio2017-workload-vctools v1.3.3
- chocolatey-visualstudio.extension v1.9.0
- vccredist2015 v14.0.24215.20170201
- dotnetfx v4.8.0.20190930
- kb3035131 v1.0.3
- python v3.9.7
Packages requiring reboot:
- vccredist140 (exit code 3010)
The recent package changes indicate a reboot is necessary.
Please reboot at your earliest convenience.
Type ENTER to exit:
```

Figure 1.20 – Windows Node.js installation: additional tools

## NOTE

Chocolatey is a package manager for Microsoft’s Windows operating system. If you are familiar with the macOS environment, this would be similar to Homebrew or Apt on a Debian Linux distribution. For more information on Chocolatey, please refer to the following link: <https://chocolatey.org/>.

## macOS

You can install Node.js for macOS via its package image, which is located at <https://nodejs.org/en/download/>, or you can install it with Homebrew by running the following command:

```
brew install node@16
```

To confirm that your machine is using the correct “node” binary, we can always check the version by running the following command:

```
node -v
```

## Linux

For Ubuntu/Debian Linux distributions, we can use a specific repository to install Node.js 14, as illustrated in the following code snippet:

```
sudo apt update
curl -sL https://deb.nodesource.com/setup_14.x | sudo
bash -
```

After the repository has been added, we can install Node.js and check the version, like so:

```
sudo apt -y install nodejs
node -v
```

So far, we have finished installing MySQL as our DBMS, applicable package managers, and the Node.js runtime library; we can now begin to scaffold our project and install the necessary Node.js packages for Sequelize and Express.

## Configuring Sequelize within an Express application

After we have installed our development tools and database, we can begin installing and configuring our application with Sequelize and **Express**. Express is a minimal web framework for Node.js runtime applications. Our Node.js application will use Sequelize to correspond with the database, and Express will relay those query results to the browser. More information on Express, along with a complete reference, can be found here: <https://expressjs.com>.

Within Command Prompt, PowerShell, or the terminal, enter the following commands for initializing our project:

```
mkdir airline
cd airline
```

```
npm init -y
```

This will create a directory called **airline**; then, we will change our working directory to the **airline** folder, and we will run an initialization script from the **node package manager (npm)**. The **npm** command will create a **package.json** file that contains a bare configuration for npm to use on this project. After that, we will need to install the minimum required Node.js modules for our application, as follows:

```
npm install express sequelize mysql2
```

Here is an online resource that you may refer to for a complete list of options for npm:

<https://docs.npmjs.com/cli/v7/commands>

Sequelize has a companion executable to help us initialize our project, manage updates to our schema, and handle database migrations. We can install it as a global (**--location=global**) binary within our userspace by entering the following command in our terminal:

```
npm install --location=global sequelize
```

For a full list of commands available to you, the CLI has documentation built in that can be exposed using the **-h** or **--help** flags, as illustrated in the following screenshot:

```
MacBook-Pro-4 airline → npm install -g sequelize-cli
/usr/local/bin/sequelize-cli -> /usr/local/lib/node_modules/sequelize-cli/lib/sequelize
/usr/local/bin/sequelize -> /usr/local/lib/node_modules/sequelize-cli/lib/sequelize
+ sequelize-cli@6.2.0
updated 1 package in 3.062s
MacBook-Pro-4 airline → sequelize-cli -h

Sequelize CLI [Node: 14.17.5, CLI: 6.2.0, ORM: 6.6.5]

sequelize-cli <command>

Commands:
  sequelize-cli db:migrate                Run pending migrations
  sequelize-cli db:migrate:schema:timestamps:add  Update migration table to have timestamps
  sequelize-cli db:migrate:status          List the status of all migrations
  sequelize-cli db:migrate:undo            Reverts a migration
  sequelize-cli db:migrate:undo:all        Revert all migrations ran
  sequelize-cli db:seed                   Run specified seeder
  sequelize-cli db:seed:undo              Deletes data from the database
  sequelize-cli db:seed:all               Run every seeder
  sequelize-cli db:seed:undo:all          Deletes data from the database
  sequelize-cli db:create                 Create database specified by configuration
  sequelize-cli db:drop                   Drop database specified by configuration
  sequelize-cli init                      Initializes project
  sequelize-cli init:config               Initializes configuration
  sequelize-cli init:migrations           Initializes migrations
  sequelize-cli init:models               Initializes models
  sequelize-cli init:seeders              Initializes seeders
  sequelize-cli migration:generate         Generates a new migration file [aliases: migration:create]
  sequelize-cli model:generate             Generates a model and its migration [aliases: model:create]
  sequelize-cli seed:generate              Generates a new seed file [aliases: seed:create]

Options:
  --version  Show version number [boolean]
  --help     Show help [boolean]

Please specify a command
MacBook-Pro-4 airline →
```

Figure 1.21 – Sequelize CLI installation and help guide

The next step is to initialize a generic template that Sequelize provides for us from the CLI. This will generate several directories for configuration, migration, seed, and model files. Here’s the code to do this:

```
sequelize init
```

The following list offers a brief explanation of the directories created by the CLI in our project’s directory:

- **config:** A directory that contains a database connection configuration file in **JavaScript Object Notation (JSON)** format. The **sequelize-cli** tool uses this configuration file to migrate schema and data files, but these configuration settings could also be used for our Node.js application as well.
- **migrations:** A directory containing Node.js files with instructions for Sequelize on how to scaffold your database’s schema and structure.
- **models:** A collection of Node.js files with Sequelize schema definitions.

- **seeders**: Similar to the **migrations** directory but instead of defining our database's schema, we will define our database's data.

Now that we have the initial foundation of our application, we can edit our Sequelize configuration file located in **config/config.json**. Depending on which installation instructions you followed, the username and password values may be different than the book's code base. The code is illustrated in the following snippet:

```
{
  "development": {
    "username": "root",
    "password": null,
    "database": "airline",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": null,
    "database": "airline",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
    "password": null,
    "database": "airline",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

If you do not wish to keep usernames and passwords in a file (which is a good idea for production environments or version control repositories), there is an alternative form for the configuration file that can accept an environment key with a connection **Uniform Resource Identifier (URI)** resource as the input (for example,

`mysql://root:password@127.0.0.1:3306/airline`), as illustrated in the following code snippet:

```
{
  "development": {
    "use_env_variable": "DB_DEV_ENV"
  },
  "test": {
    "use_env_variable": "DB_TEST_ENV"
  },
  "production": {
    "use_env_variable": "DB_PRODUCTION_ENV"
  }
}
```

If we wanted to use the **development** configuration, our Node.js application would know to look for the connection parameters/URI from an environment variable called **DB\_DEV\_ENV** (you may use the same environment variable for any stage). For more options and configuration settings for the Sequelize CLI, refer to this resource:

<https://github.com/sequelize/cli/blob/master/docs/README.md>.

## NOTE

You can toggle between which environment you would like your application to be in by setting a **NODE\_ENV** environment variable. The default value is **development** but if we wanted to use our **production** environment, we would set the environment like so:  
**NODE\_ENV=production.**

## Connecting Sequelize with Express

We can now begin building our Node.js application by creating an **index.js** file within the project's directory and opening the file in our IDE of choice. Let us begin by typing in the following code:



```
const express = require("express");
const app = express();
const models = require("./models");
models.sequelize.sync().then(function () {
    console.log("> database has been synced");
}).catch(function (err) {
    console.log(" > there was an issue synchronizing the
                database", err);
});
app.get('/', function (req, res) {
    res.send("Welcome to Avalon Airlines!");
});
app.listen(3000, function () {
    console.log("> express server has started");
});
```

We begin by declaring our Express/web application variables (**express** and **app**) with the first two lines of the code. The next line is shorthand for invoking the `./models/index.js` file that was created by the Sequelize CLI from earlier (we will go into details of that file in the next chapter). The following line runs the Sequelize **sync()** command, which will synchronize your model definitions with a database by creating the necessary tables, indices, and so on. It will also establish associations/relations, execute sync-related hooks/events, and so on.

The **sync()** command offers several options that are encapsulated within an object as the first parameter, as outlined here:

- **force**: A Boolean value that will drop your database's tables before re-creating them.
- **match**: A **regular expression (regex)** value to match against table names to sync. Useful for testing or to ensure only certain tables are affected by the **force** option within a production environment.
- **logging**: A Boolean or function value. **true** (the default) will use **console.log** when executing queries for logging. **false** will disable entirely, and a function can be used to send logs and context to another



adapter. This book will go into detail about this option in a later chapter.

- **schema**: A string value for defining which database to operate in. Useful for when you are using a DBMS such as Postgres, which allows you to separate tables by not only a database (which MySQL calls a “schema”) but also by a namespace (which Postgres calls “schema”).
- **searchPath**: A string value to define the default **search\_path** for Postgres databases only. This option will not pertain to this book’s code base or content.
- **hooks**: A Boolean value (defaults to **true**) to execute several hooks/events that are related to sync events (**beforeSync**, **afterSync**, **beforeBulkSync**, and **afterBulkSync**). **false** will disable events from executing.
- **alter**: An object with the following parameter:
  - **drop**: A Boolean value that prevents any **drop** statements from being executed when Sequelize needs to run **ALTER** commands within the database.

You can define these options like so:

```
models.sequelize.sync({  
  force: true,  
  logging: false  
})
```

#### NOTE

It is *not* recommended by the Sequelize community to run the **force** option as **true** within a production environment. This could have unintentional consequences such as deleting vital customer/user information. The **force** option is for when you are still prototyping your application and want to start your application on a clean slate per iteration.

The next command, **app.get(...)**, instructs the Express framework to route the “/” (root) path of our web application to the scoped function (in

this case, we are sending text back to the browser, as shown in *Figure 1.22*). After that, we start the Express server by calling `app.listen(...)`, which will tell our application to listen for **Hypertext Transfer Protocol (HTTP)** events on port **3000**, which can be accessed in our browser via `http://localhost:3000` or `http://127.0.0.1:3000`, depending on your network interface settings. For starting our application, we can run the following command within our terminal/PowerShell:

```
node index.js
```

You should see text displayed on your screen indicating the following:

- Express has started
- A SQL query was executed
- The database has been synced

#### NOTE

Sequelize will automatically execute a `SELECT 1+1 AS result` query as a method for checking on the database connection's health. Not all DBMSs offer a way of sending a **ping** packet to check whether a connection is successful or not.

Now, when you open your browser and visit the previously mentioned URL, you should see a page similar to what is shown here:

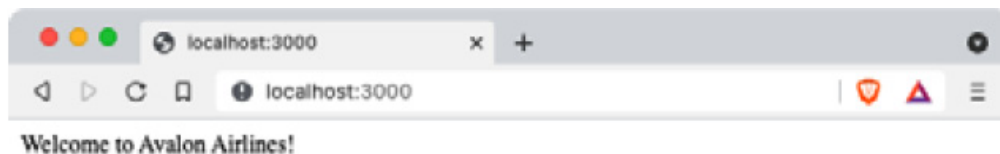


Figure 1.22 – Welcome page

Every time we make a change to our application, we will need to terminate our current process (`Ctrl + C`) within the terminal. This will send a **SIGINT** signal to the process, which will send an interrupt signal to the process in order to begin cleaning up and then exit/stop. To avoid having to restart our process manually after every change, we can install a sepa-

rate process to help facilitate this for us called Nodemon (more information can be found here: <https://nodemon.io/>).

Nodemon may be installed as a global binary by running the following command:

```
npm install -g nodemon
```

You can confirm if the installation was successful by typing in the following:

```
nodemon index.js
```

This should start our Node.js application while simultaneously watching for changed files within our project's directory. Once we have made a modification to the project, we should see Nodemon automatically restarting our process, as illustrated in the following screenshot:

A screenshot of a terminal window on a MacBook-Pro-4. The terminal shows the command 'nodemon index.js' being executed. The output indicates that Nodemon (version 2.0.12) is watching for changes in the current directory and its subdirectories for files with extensions .js, .mjs, and .json. It starts the application 'node index.js', which runs an Express server. The server logs show 'express server has started', 'Executing (default): SELECT 1+1 AS result', and 'database has been synced'. After a few moments, Nodemon detects changes and logs '[nodemon] restarting due to changes...' followed by '[nodemon] starting `node index.js`'. The application then restarts, showing the same logs again. The terminal window has a title bar with standard macOS window controls and a status bar at the bottom showing system metrics like memory usage (0.0 kB down, 1.0 kB up), disk usage (10%), and available space (24 GB).

```
node
0.0 kB↓ 1.0 kB↑ 10% 24 GB ~/airline
λ MacBook-Pro-4 airline → nodemon index.js
[nodemon] 2.0.12
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
> express server has started
Executing (default): SELECT 1+1 AS result
> database has been synced
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
> express server has started
Executing (default): SELECT 1+1 AS result
> database has been synced
```

Figure 1.23 – Nodemon automatically restarting the application

The last step for this chapter is to make a few adjustments to our `package.json` file, as follows:

- Add `"private": true` under the `"name": "airline,"` line. This adjustment will prevent us (or anyone else on the team) from publishing our project to the public npm registry.
- Look for the `scripts` object and replace whatever content is there with `"start": "nodemon index.js"`. This will allow us to start our applica-

tion by running the following command:

```
npm run start
```

The final `package.json` file should look similar to this:

```
{
  "name": "airline",
  "private": true,
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "mysql2": "^2.3.0",
    "sequelize": "^6.6.5"
  }
}
```

## Summary

In this chapter, we introduced the benefits of using an ORM and what Sequelize has to offer. We learned how to set up our development/local environment to run a DBMS (MySQL) and the Node.js runtime. We then scaffolded a project using npm and the Sequelize CLI and integrated the Sequelize library with the Express web framework.

In the next chapter, we will begin inserting data into our database and define Sequelize models.

[Support](#) | [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) | [PRIVACY POLICY](#)