

# THE NEWBIE'S INTRODUCTION TO CAKEPHP

*By David Golding*

*This guide does not represent the practices, ideas, or conventions of the CakePHP Software Foundation and is not an official publication of the CakePHP community at large. Any suggestions, tutorials, or instructions contained in this book are statements of the author alone and should not be construed as an official comment of the developers of CakePHP or the other technologies cited. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the author. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the author assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.*

*Book is copyright (©) 2007, David Golding.*

# TABLE OF CONTENTS

<b>Introduction.....</b>	<b>5</b>
<b>Why Cake.....</b>	<b>5</b>
Models, Views, and Controllers (MVC).....	5
CRUD Operations.....	6
Scaffolding.....	6
Helpers.....	6
Large Community.....	6
Much More.....	7
<b>A Newbie's Guide.....</b>	<b>7</b>
What This Book Is Not.....	7
<b>Installing Cake.....</b>	<b>8</b>
<b>Localhost First, Remote Last.....</b>	<b>8</b>
Why Doing It All Remotely Is Bad.....	8
Localhost Setup.....	9
Setting Up on a Mac.....	9
Setting Up on a PC Running Windows.....	11
<b>Running MySQL.....</b>	<b>13</b>
A Quick Overview of Database Management.....	13
How It's Organized.....	14

PHPMyAdmin.....	15
Other MySQL Applications.....	15
Typical Settings.....	16
<b>Running CakePHP.....</b>	<b>17</b>
Off the Ground.....	17
Setup Routines.....	18
<b>Your First Cake App.....</b>	<b>23</b>
<b>Plain Text Editors and Other Applications.....</b>	<b>23</b>
Plain Text Editors.....	23
Others.....	25
<b>A To-Do List Application.....</b>	<b>25</b>
Create the Database Schema.....	25
Create the Items Controller.....	26
Create the Items Model.....	27
Launch Your App.....	28
<b>Naming Conventions.....</b>	<b>28</b>
Controllers.....	29
Models.....	29
Views.....	29
More Than One Word in the Title.....	29
<b>Changing the Design.....</b>	<b>30</b>

How Views Work.....	30
<b>Creating Individual Views.....</b>	<b>33</b>
Add the Index Action to the Controller.....	34
Create the Index View.....	35
<b>Using Bake to Create Views.....</b>	<b>37</b>
Getting the Bake Script Working In Our Localhost.....	37
Bake the CRUD Views.....	39
<b>Cleaning Up the Views.....</b>	<b>42</b>
Make the Priority Field a <select> Tag.....	43
Change the Index View to Better Display the Completed Field.....	43
<b>Important Points.....</b>	<b>44</b>
<b>Level Two Applications.....</b>	<b>46</b>
<b>A (More Extensive) Blog Application.....</b>	<b>46</b>
Database First, Always.....	46
Table Associations.....	47

# INTRODUCTION

## *Why Cake*

Ever since the boys at 37 Signals ([www.37signals.com](http://www.37signals.com)) released Ruby on Rails as an open source project, the world of web development has been moving more toward a framework-based solution for creating dynamic websites. Gone are the days of individually scripting each and every function the website will utilize. Instead, developers are using a bundled package of scripts that perform a series of functions designed specifically for web development.

Because Rails set the standard for web-based frameworks, teams of developers have been creating clones of Rails for various languages: Turbogears for Python; Symfony, PHP on Trax, Zend and many others for PHP; and Catalyst for Perl—and on and on. With so many options out there, why choose CakePHP (or, for short, Cake) for your web project?

## *Models, Views, and Controllers (MVC)*

Cake uses the MVC structure as it compiles. What this means is that it effectively separates typical operations into specific areas: models for all your database interaction; views for all your output and displays; and controllers for all your commands/scripts.

MVC can vary depending on the framework with which you're working. But generally it works like this.

1. The user interacts with the interface in some way (e.g., presses a button).
2. The controller handles the input event.
3. The controller accesses the model to fetch any saved data or fetch data stored in the database.
4. The view uses data handled by the controller and model to generate the user interface or any other output that must be sent to the user.
5. The interface waits for further user interaction before the cycle begins anew.

The benefit of using MVC structures in developing websites is that repeated functions or tasks can be separated, thus allowing for quicker edits. It can even help in debugging. Say an error keeps occurring during the interaction with the database. Usually the problem will be somewhere in a model, or least likely in the controller (if you follow convention). Knowing that all database interactions occur in just one place makes addressing problems more efficient.

Cake makes MVC developing a snap and actually has a straightforward folder structure that cuts down on nested folders and extra directories. When deciding between other frameworks, realize

that Cake has a more efficient MVC structure than Symfony or Zend and will occupy less space on your server.

## *CRUD Operations*

Most websites make use of CRUD operations: create, read, update, and delete. A blog, for example, will need to create posts; users will need to be able to read each post; the author will likely want the ability to edit the post in the future or update the post; and the author will also want access for deleting posts.

Cake makes CRUD operations a breeze with automated CRUD functions. Instead of writing by hand each CRUD operation, Cake has pre-built classes that do it for you.

## *Scaffolding*

Getting a website off the ground is much easier with Cake's scaffolding abilities. With just one simple line of code, you can call out Cake's pre-built scaffold to render views based on the database. In other words, it figures out how some standard interface views should work with your database, and outputs the HTML forms and all without you having to write one bit of HTML. While the scaffolding feature is not intended for full production views, it allows for you, the developer, to begin testing logic and functions without wasting time building views or HTML output.

## *Helpers*

Cake comes with standard HTML, Ajax, and Javascript helpers that make creating views much easier. Your HTML output will be greatly facilitated by coherent strings of helper code that render the markup for you. And getting Ajax to work, while a little tricky at first, is much easier and far more efficient than building those nasty Ajax scripts yourself. What's more, you can download other helpers written by fellow Cake developers to boost the strength of the framework, or even write your own to cut down on repetitive or clumsy markup.

## *Large Community*

Should you need help down the road, a massive online community exists to help out. Not only can you appeal to expert advice from Cake gurus on forums and message boards, but even more PHP experts can help you with learning the code. In reality, the PHP community is the largest programming group on the web, so if you need a quick workaround for a problem in Cake, someone somewhere will have some help for you, usually within minutes.

Code samples are a must for anyone getting involved in web development. PHP dominates this field, and Cake has a growing repository of code samples as well. When considering going with another framework, this fact just may tip the scales in favor of Cake if you are wanting to piggyback on someone else's work.

## *Much More*

As you can see, Cake aims to simplify the development process through competing features that have significant advantages when learning to develop web software or taking on web development tasks. Even more features could be talked about, but since this is a newbie's guide, we'll save that for more advanced conversations. What is important to note is that Cake is so far the most popular PHP web framework, and it has a great start on providing needed features for PHP developers that you can't pass up.

## ***A Newbie's Guide***

This guide is for true newbies to PHP and Cake. If you're already familiar with Ruby on Rails or another web framework then this guide may be a little redundant for you. But if you don't know what a "has-and-belongs-to-many" relationship is or how to parse an array, then this just may be where you want to start when getting into Cake.

After poring over the online documentation, I realized there wasn't much by way of an introduction to Cake. Most of the available resources require some sort of prior knowledge of web development to get a grasp on how to install and work in Cake. If you're like me, you probably just want a handful of tutorials with code samples from square one that can get you up and running quickly and lead you in the right direction for more advanced techniques.

Furthermore, when asking a question on forums or chat rooms, many newbies get little help from the experts. Simple questions can get a response like, "Well, just read the online manual and API." Sometimes we newbies need a very simple approach to the software, and this guide will do just that. We'll start with installing Cake on a server and a localhost, what it all means, and provide some detailed code samples and visual snapshots to walk you through. By the end of the book, you ought to have a solid-enough foundation to give other advanced features a try on your own.

## *What This Book Is Not*

Now, I recognize that there are myriad scenarios for installing PHP, MySQL, CakePHP, Apache, and more. I won't explore every possible setup. Rather, this book will walk through *one* method for working with Cake with the newbie in mind. So my aim is to go about the most simple method for working with web servers and software. If you are searching for a unique way for working in Cake, you ought to consult the web in general, the Google group, or other experts on the Cake chatroom on IRC (all this can be found by going to [www.cakephp.org](http://www.cakephp.org)). If, however, you're stupified over getting Cake, PHP, MySQL, Apache, and other web technologies to work, then this book will get you started off well.

# INSTALLING CAKE

## *Localhost First, Remote Last*

The best way to setup your development environment is to configure a localhost server on your computer. Let me explain what that means. When you access a website, you're sending and requesting information from another computer through an internet connection. That server can communicate and process the information in a plethora of possible configurations. For example, when you buy a computer, you have the option of getting a Mac or Windows-based machine. Or, if you so wish, you could run Linux, Unix, or even DOS. All these possibilities mean that for you, the user, your experience will depend on which operating system you choose. But let's say you want to print a document. Well, regardless of the OS, or how that OS prints something, once the document is printed, it doesn't really matter what configuration the computer had, it's all the same to the reader of the document.

The internet is a lot like this example. The computer/server at the other end that will be processing the requests and creating output could run just about anything. But the final output will be a web page of some sort, or some kind of web output. For you as the developer, the configuration will matter a great deal. In fact, it will affect everything that you do to create and serve a web site.

When you set up a localhost, you're actually setting up a server on your own computer. You'll fire up a web browser like Firefox or Safari and type in something like "http://localhost" to access the server like the user would if the web site were live. In effect, you're tricking the computer into thinking that it's talking to a web server when really it's talking to itself.

As long as your configurations on your localhost are the same as the configurations of the remote web server where the site will be hosted, you can develop on your own computer. Once the site is running well, you'll move the entire contents of your root directory or application to the server, and you're done. If you've done it right, the site should run all the same on the remote server.

## *Why Doing It All Remotely Is Bad*

Should you circumvent the localhost setup, you'll be forced into developing everything remotely. That's fine, I suppose, if the following things don't bother you:

- Running a continuous FTP session.
- Uploading every time you want to test the slightest alteration to your code.
- Timeouts due to your internet connection occurring during program executions.
- Creating alternate names to test in environments of which no one is aware.
- Users accidentally stumbling on your half-baked programs.
- Eating into your own bandwidth (which for some hosting plans may be tight as it is).



## Localhost Setup

Whatever OS you're running on your computer, installing a localhost environment is easy. If running on a Mac, I recommend installing Living-E's *MAMP*. If on a Windows-based PC, install *XAMPP*.

Normally, creating a localhost setup will require installing various server programs via a command line interface that takes a good understanding of PHP, MySQL, Apache, ProFTPD, and other server technologies. What MAMP and XAMPP do is simplify these complex server setups so that essentially you run an installer program and the setup is complete.

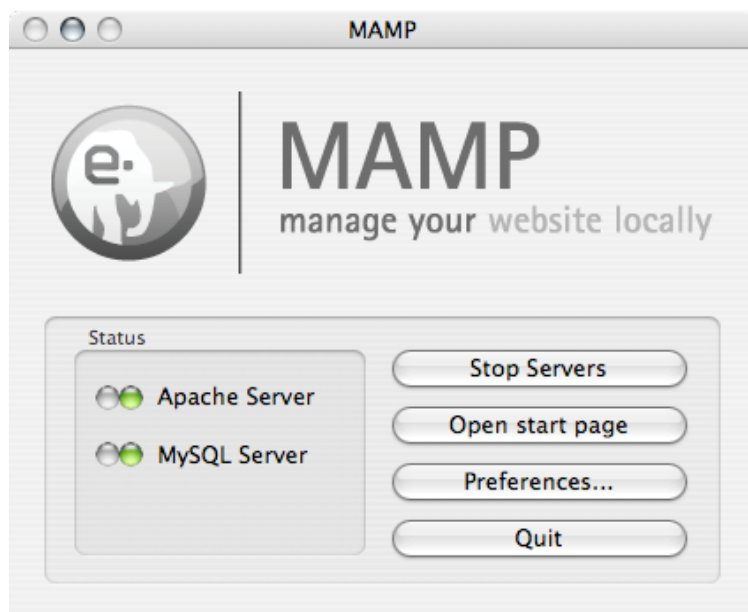
### Setting Up on a Mac

Mac OS X already comes with a localhost setup built in. But if you want to run MySQL and upgrade PHP to version 5.0 or higher, you'll need to install each element manually, which can be a little tedious. Running MAMP will likely save you a lot of time. Here's how to do it.

**Download the software.** Go to <http://www.mamp.info> and locate on the home page the MAMP download link. It will ask you if you want MAMP Pro, and if you do, go ahead and pay the \$49 to download it. Otherwise, download the latest free version of MAMP.

**Open the disk image and install the MAMP folder.** This step is simple enough. It will ask you to drop a folder into the Applications folder. This is necessary for the localhost setup to work properly. Make sure you do not use any subfolders or rename the MAMP folder to anything else. If you dislike having to do this, you may want to consider more elaborate localhost setup methods.

**Run the MAMP application program.** At any time you can manage your localhost setup by running the application *Applications > MAMP > MAMP*. This main screen looks like this:



This program, when you click “Start Servers,” will maintain the localhost environment. You now can begin executing PHP code, but you have to know where to access the localhost on your hard drive.

**Configure the localhost root folder.** By default, any file contained in the *Applications > MAMP > htdocs* folder can be accessed in the localhost environment. This isn’t too bad, altogether, but I personally prefer having my web sites outside my Applications folder. Changing this setting is as simple as going into the preferences area of the MAMP application.

Click on “Preferences...” to bring up the preferences area. Click the “Apache” tab, and you’ll find a field called “Document Root.” It should read “/applications/mamp/htdocs” or something like that. Click “Select...” to change the document root to something else. In this case, I’m going to select the *~/Sites/* folder (if my Mac OS X user name is “dave,” then the path will look like */Users/dave/Sites*). Now, from here on out, any folder or file places in the Sites folder will be executed by the MAMP localhost setup and not Mac OS X’s default web sharing.

**Change the localhost ports.** Now click on the “Ports” tab, still in the Preferences area. It lists the Apache port and the MySQL port. The values in these fields are set to 8888 and 8889, respectively, by default. To better mimick a typical remote web server, these should be changed. In the Apache port field, type “80” and in the MySQL port field, type “3306.” (You can also click the button “Set to default Apache and MySQL ports.”)

Because these ports are configured this way, you can now open Safari and type “http://localhost” instead of “http://localhost:8888” whenever you want to execute scripts and files. Also, when configuring programs to work with MySQL, you can simply enter “localhost” in the server setting instead of “localhost:8889.”

**Make the localhost easier to manage.** It can’t get much easier than this to get a localhost environment running on a Mac. But one flaw with the MAMP application is that it *must* be running for the localhost environment to be running. So if you quit out of MAMP, you will no longer be able to run PHP code via the localhost. Fortunately, Living-E provides a Dashboard widget that does exactly what the MAMP app does to keep the localhost active. Simply launch the “Mamp Control.wdgt” file and click “Start Servers” to make the localhost active. It will now run in the background through Dashboard.



**“Hello world!”** Let’s make a “Hello world!” script to make sure our localhost is working right. Go to the root folder (again, mine is located in my “Sites” folder) and create a new folder called “hello.” In that folder, create a plain-text file called “index.php.” In index.php, paste the following code:

```
1 <?
2     echo "Hello world!";
3 ?>
```

Now, open up your web browser and type “http://localhost/hello.” You should get “Hello world!” to display on the screen. You are ready to begin an installation of CakePHP.

## Setting Up on a PC Running Windows

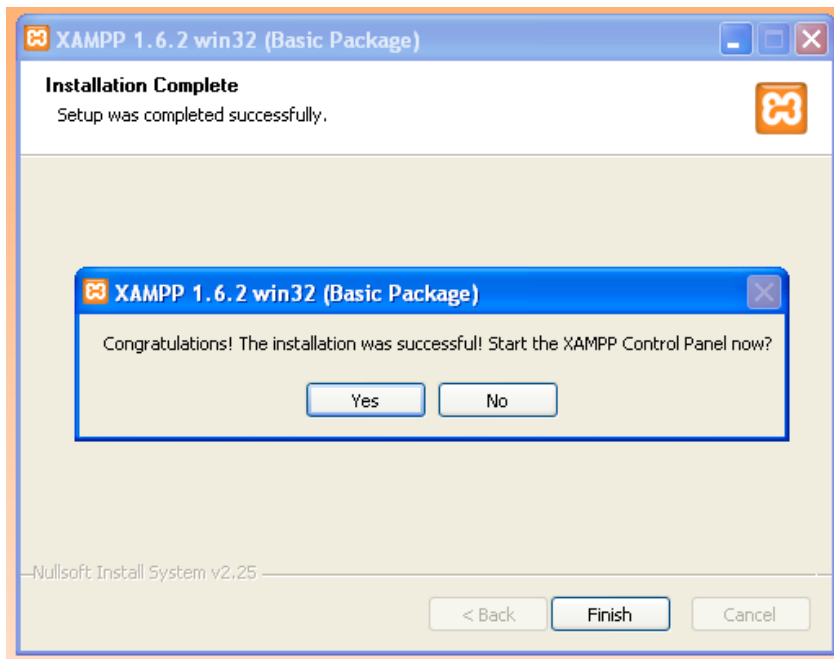
Windows XP Pro has Microsoft's web server built into it, known as "Internet Information Server" (IIS). To install IIS is pretty easy to do. However, most PHP/MySQL scenarios are more easily handled in a bundled localhost installer available from Apache Friends ([www.apachefriends.org](http://www.apachefriends.org)) called XAMPP. Here are the steps to take when installing and configuring XAMPP on your PC.

**Download the software.** XAMPP can be downloaded from the Apache Friends website. As of the printing of this book, a direct link is right here:

<http://www.apachefriends.org/en/xampp-windows.html>

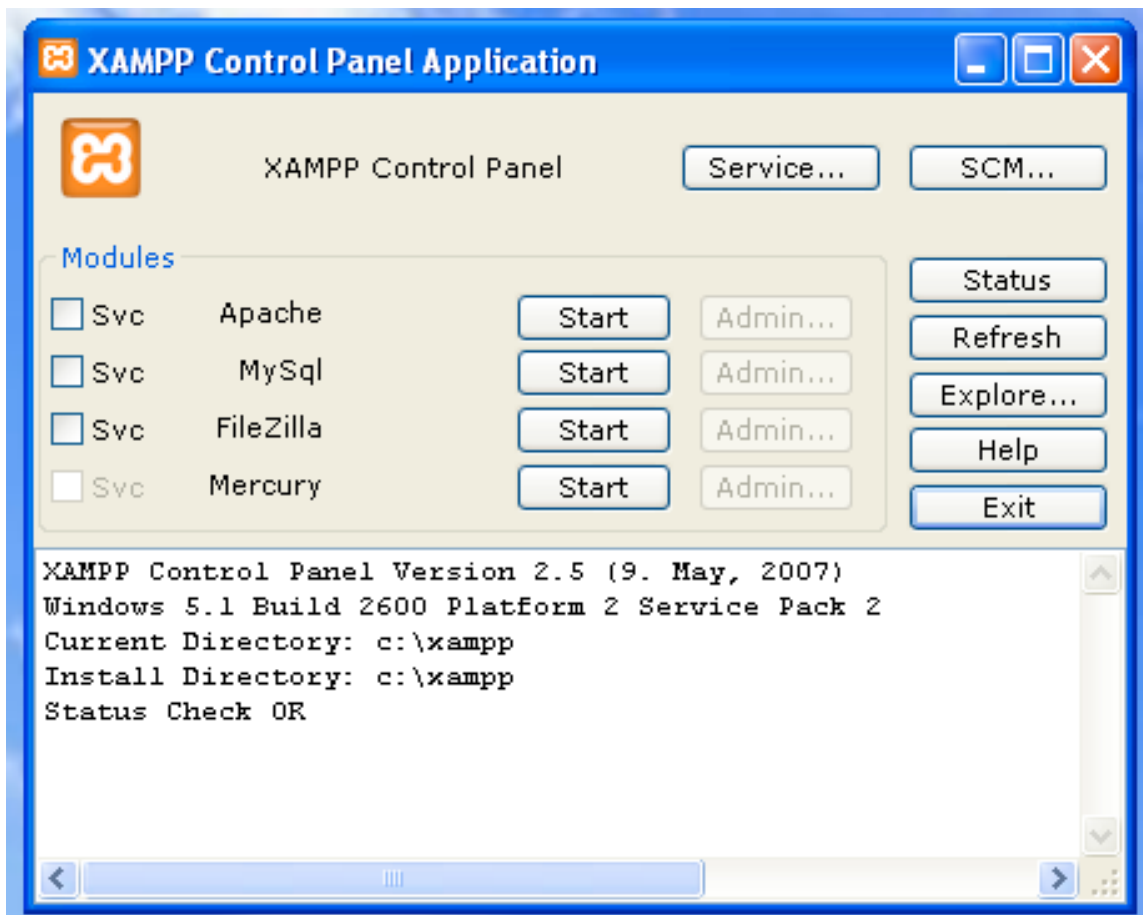
You can download an installer application, a Zip file, or a self-extracting 7-ZIP archive. The easiest option is to just run the installer application. So scroll down to the download area of the page and click "Installer." The download should immediately commence.

**Run the installer program.** The program will walk you through installing XAMPP. It will ask for a destination folder for XAMPP, and the default location is at `c:\xampp`. I recommend sticking to this configuration since you will likely want XAMPP to be able to access any area of your C-drive. Click "Next," and on the next screen leave the "Service Section" checkboxes blank. To wrap it all up, click the "Install" button at the end. It will extract all the necessary files and leave you with a convenient localhost environment in the `c:\xampp` folder.



**Open the XAMPP Control Panel.** After installation is complete, you'll be taken to the XAMPP Control Panel. This panel is accessible in the future by clicking on the shortcut icon on the desktop, or by going to *Start > Apache Friends > XAMPP > XAMPP Control Panel*.

The screen looks something like this:



Clicking the start buttons starts those important web server services like Apache, MySQL, and more. Start up Apache and MySQL using the start buttons and fire up your web browser. In the URL field, type “http://localhost”. You’ll be taken to the XAMPP welcome screen.



[English](#) / [Deutsch](#) / [Français](#) / [Nederlands](#) / [Polski](#) / [Italiano](#) / [Norsk](#) / [Español](#) / [中文](#) / [Português \(Brasil\)](#) / [日本語](#)

**Make the localhost more secure.** Unfortunately XAMPP doesn't come pre-installed with a more secure localhost environment. In some ways this may be a good thing, but for right now, the last thing you want is to compromise your network or your computer. On the left side of the XAMPP startup web page is a link to "Security." Go ahead and click that, bringing up the security information page. Most items, if not all, will be marked "unsecure." Let's make those all light up green.

A link below the main table called "<http://localhost/security/xamppsecurity.php>" takes you to some fields that will allow you to secure the localhost. Change the MySQL root password and click "Password changing". Below the MySQL root password fields is another set of fields to protect the XAMPP directory (.htaccess). Put in a user name and password there. For both, make sure you *do not* save the passwords in a plain text file.

From here on out, you will be asked to log in to access the XAMPP control pages. Be sure not to lose your user names and passwords! (I really shouldn't have to say that, but you never know.)

Restart Apache and MySQL to propagate the settings you've created.

**"Hello world!"** Let's make a "Hello world!" script to make sure our localhost is working right. Go to the root folder (located now at `c:\xampp\htdocs`) and create a new folder called "hello." In that folder, create a plain-text file called "index.php." In index.php, paste the following code:

```
1 <?
2     echo "Hello world!";
3 ?>
```

Now, open up your web browser and type "<http://localhost/hello>". You should get "Hello world!" to display on the screen. You are ready to begin an installation of CakePHP.

## REMINDER

From here on out whenever I'm referring to your "root" folder, it's going to be where the localhost can access scripts and files. For Mac users who followed this tutorial, this folder will be the user's "Sites" folder. For Windows users running XAMPP, this folder is the "htdocs" folder in the `c:\xampp` folder.

## Running MySQL

MySQL is an open source database program that runs well on web servers. In your localhost set-ups MySQL is already installed, but we have to now access the program to begin creating databases for Cake to use. First, a note on database management.

### *A Quick Overview of Database Management*

Many web sites run on plain files without any use of databases or dynamic scripting. Databases can save the day because they can hold data for easy retrieval on your server. All those accounts people create to use big-name sites like Ebay or Amazon are stored in some kind of database, not

in plain text files. While this can be extremely useful, all the talking back and forth between your scripts and the database can result in lines and lines of query strings and code.

SQL is a light-weight language (it stands for “structured query language”) that makes data retrieval much less complicated than other high-end database systems. There are extensions that add more functionality to SQL, depending on the aim of the extension. Actually, SQL is only a language and nothing else—it isn’t an application itself but it can be applied to any database program out there. SQL extensions sure make life a lot easier for developers in that they simplify the setup, connecting, scripting, and maintaining of SQL database servers and tables.

ANSI, IBM, Microsoft, and Oracle all extend SQL in different ways. MySQL is just an open source extension that is designed with the web in mind. I’ve chosen MySQL for use in this book because it is *far* more common among web hosting providers and PHP programmers than the others. It’s also got a lot of community support out there, so if you are really itching to get a deeper look into MySQL, it’s only a couple clicks away. As you are a newbie to CakePHP, I’m going to assume that you’re a newbie to MySQL as well.

### *How It’s Organized*

Whether you’re using one MySQL application or another, MySQL has the same overall structure. Here we’ll go over the different parts of MySQL and how they relate to one another to provide you with efficient database management tools.

**Databases (or, sometimes called Schemas).** MySQL can store multiple databases. In theory, a database is going to be dedicated to one web site or web application a piece. So if I were to create a shopping cart application, I might call my database “cart.” This database would connect to my shopping cart scripts alone and store all the necessary information for that one application to work. If I had another email application on my site, I might put all that necessary information into another database called “email.” The more individual my applications are (e.g., “cart” and “email” have little to do with each other as users interact with them on the site), the more ideal it is to keep these applications to their own databases. Rarely, you may want your databases to fetch data from a separate database. But most of the time, you’ll want to have your scripts fetch data from just one database.

**Tables.** Each database contains one or more tables. A table is a set of records grouped together. The necessary fields and cells are organized by the table, so this is usually the first step in building your database. When creating a table, MySQL asks you how many fields (columns) you’ll want to create and how to name them. Going back to my shopping cart example, I’ll probably want to have a table called “orders.” In this table, each entry would represent one order. So, I’ll ask myself “What does each order have in common?” A person who placed the order, an item ordered, an amount paid, and so forth. These elements make up the fields of the database table and are filled up with data when a record is created.

**Relational tables.** Tables can relate to one another in complex ways. In a shopping cart program, not only will users store information regarding their orders, but they may select items from a catalog to buy. I’d then build that catalog as a table in the “cart” database and call it “items.” Each

record in the items table would be one product with a price, shipping cost, etc. Then the “orders” table would have a space for linking up with the “items” table, because each order would be for at least one item. We’ll go into detail about relational tables and databases; for now, know that tables can be linked to one another to simplify data retrieval and organization.

**Records and fields.** As explained in the tables examples, each table contains one or more records (rows) of data. It’s best to build your tables with records in mind. If, for example, the table is mainly built for “user accounts” then each record ought to represent one user account. If the scope of your database gets more complicated than this, then it’s best to build relational tables and link them to one another. This cuts down on scripting as well as load time.

Fields are the areas each record shares that are specified by the table. An array of available field options exist, depending on the type of field (a name versus a date or checkbox) and how much data the field will hold (12 characters versus 200). Each field will always be assigned a type, length, and default value (which can be blank, if you want).

## *PHPMyAdmin*

The most powerful web-based tool for setting up and running MySQL is called *PHPMyAdmin*, another open source application that comes installed with MAMP and XAMPP. The startup screens for both localhost environments contain a link to PHPMyAdmin. This is the quickest way to begin working in MySQL, since we’ve already installed a localhost that has the program pre-configured and ready to go. A great many developers prefer PHPMyAdmin over other desktop-based MySQL applications because it’s web-based and free, and quite honestly, a great application. I prefer desktop-based applications because of their speed and I can avoid screen refreshing which PHPMyAdmin is locked into for being a web application.

## *Other MySQL Applications*

If you’d like to use a desktop application to manage your databases, here are a couple great sources to try out. They will likely require an extra level of configuration to get them working with your localhost environment. I’ll list them below then explain some of the necessary settings to get them working right.

**CocoaMySQL.** On the Mac, probably the simplest freeware application for running MySQL is CocoaMySQL. (It’s available by going to <http://cocoamysql.sourceforge.net>.)

**MySQL Query Browser.** On both Mac and PC, MySQL Query Browser is a reliable and easy freeware application that lets you navigate MySQL through a graphical interface (<http://dev.mysql.com/downloads/gui-tools>).

**HeidiSQL.** Much like CocoaMySQL on the Mac, except built for Windows. The layout is, in some ways, much easier to navigate than MySQL Query Browser and will require a less MySQL prowess to get started. It, too, is open source, so giving it a try can’t hurt ([www.heidisql.com](http://www.heidisql.com)).

## Typical Settings

No matter what application you decide to use, each one will need some parameters before it can connect to MySQL. Here are the common ones that get you connected.

- Host
- User
- Password
- Socket
- Port

**Host.** This will almost always be just “localhost,” even when you’re setting up a script to run on your remote server. If you are trying to access a remote database, it might be the domain name of the site, but you’ll need to have the details provided from the web hosting provider. With the configurations outlined above, simply enter “localhost.” Should this not work, just identify the IP address of your computer, which is usually defaulted to “127.0.0.1”.

**User and Password.** MySQL supports multiple users and groups for administering and maintaining databases. These can be added or changed once you’ve logged into your application and access the necessary areas to add or change an account. In our example, this will be both “root” for username and password, unless you entered something different instead. Sometimes web hosts will set up shared hosting accounts for you; in these cases you may need to add a prefix to the username, like “cake\_”.

**Socket.** When MySQL is running, the connecting point for the application and PHP is called the “socket.” The default path for this is `/var/mysql/mysql.sock` but with our quick and easy localhost setup configurations, this will be a dead end. If you know that the MySQL socket is stored in its default location (like on most hosting setups) then you can leave this setting blank. On the localhost, we’ll have to specify the path. On the PC, the defaults connect correctly if you followed the tutorial. On a Mac, here are the path strings, if necessary.

*On MAMP:* `/applications/mamp/tmp/mysql/mysql.sock`

*On XAMPP:* `/applications/xampp/xamppfiles/var/mysql/mysql.sock`

**Port.** The default port for MySQL is 3306. Enter this only if the application asks you, or if you somehow altered the port number to something else.

### REMINDER

When referring to MySQL, I’ll assume you know how to connect to your databases, create, delete, and modify them as well. So now is a good time to try and practice creating a database, creating a table, and adding some records to that table.



## Running CakePHP

The localhost is now ready to run Cake. Download the latest stable release of Cake from [cakephp.org](http://cakephp.org). (Right now, the most current stable release is Cake 1.1. Because version 1.2 is set to officially take over, and it does have some significant improvements, we'll use 1.2 in this book.)

When you've downloaded and extracted the Cake release file, you'll end up with a folder called "cake\_1.2.0.1111" or something like that. Inside this folder contains everything you need to run Cake:

```
app/
cake/
docs/
index.php
vendors/
```

### Off the Ground

Now, let's get off the ground with this Cake stuff. It's time to actually turn the switch on.

**Rename the folder.** Name the "cake\_1.2.0.1111" folder to "cake."

**Move the cake folder.** Place the "cake" folder into your localhost root.

**Launch Cake.** Open your web browser and type "http://localhost/cake" in your URL field. You should get the following screen:



## CakePHP: the rapid development php framework

**Notice:** Please change the value of `CAKE_SESSION_STRING` in `app/config/core.php` to a sal  
`/Users/dave/Sites/todo/cake/libs/debugger.php` on line 373

Your tmp directory is writable.

Your cache is set up and initialized properly.

FileEngine is being used to cache, to change this edit `config/core.php`

Settings:

- class: FileEngine
- directory: `/Users/dave/Sites/todo/app/tmp/cache/`
- prefix: cake\_
- lock: false

Your database configuration file is NOT present.  
 Rename `config/database.php.default` to `config/database.php`

**Typical problems.** Now, you may, at this point, have encountered a couple typical problems with the installation.

*Permissions error.* There may not be the necessary file permissions in place. If this error occurs, you may have only seen a blank screen or a “403” error screen. To fix this, open the “cake” folder with “Get Info” on a Mac, or right-click and select “Properties” on a PC. You’ll need to make sure that the “System” user can read/execute the folder and all its contents. If you’re already familiar with the command-line, open a shell, browse to the folder, and give it a “chmod” value of 755. Refresh the “cake” URL and if you see the screenshot above, the problem is fixed.

*Apache AllowOverride error.* This error occurs when you see the content but it doesn’t appear like the screenshot image you see, i.e., no color, no styles, just black, default text on white. If you continue with the rest of the tutorials here, you’ll be able to see Cake running, but some things will bug in and out on you, especially the scaffolding and the styles. This fix is a little more complicated, but not hard.

You’ll need to find the “httpd.conf” file in your localhost setup. It’s usually stored in a folder named “conf,” “bin,” “lib,” or “var.” The httpd.conf file can be edited by any plain-text editor.

Search out a chunk of code that looks something like this in the httpd.conf file:

```
1 <Directory />
2     Options Indexes FollowSymLinks
3     AllowOverride None
4 </Directory>
```

Change line 3 to:

```
3     AllowOverride All
```

Then open the control panel for your localhost (either the MAMP dashboard widget or the XAMPP control panel). Restart Apache and try again to refresh the Cake program in your web browser. The problem is fixed if you see “life” brought back into the startup screen.

## Setup Routines

Every time you install a Cake application on your localhost, you’ll follow these routine procedures.

1. Prepare the “tmp” folder.
2. Set the “CAKE\_SESSION\_STRING” global variable in *app > config > core.php*.
3. Enter MySQL database connection settings.
4. Design your database schema.

**Prepare the “tmp” folder.** Inside the *root > cake > app* folder is a folder named “tmp.” Cake needs to be able to tinker with this folder to operate, so this folder’s permissions will be different than

the others. If you've used the command line before, this process is fairly simple. If not, now's the time to begin.

*Command line on the Mac.* Mac OS X comes with a built-in command line application located in the *Applications > Utilities* folder. When Mac OS X was first built, they did it on top of Unix, making the command line actually a part of the operating system. Mac users will find a basic knowledge of Unix commands extremely useful for high-end processes. Launch the Terminal application from the Utilities folder, and you're there. You type a command, hit return, and that's pretty much it.

*Command line on Windows.* You can launch the *Start > Accessories > Command Prompt* to access the command line, but this actually is running MS-DOS. For web services, we'll need to be able to run Unix commands. This will require an additional installation of command line tools.

Microsoft created its Powershell program to boost the command line functionality of Command Prompt. However, Powershell won't behave like most web developers desire; it will be missing some common commands like `ls`, `chmod`, and `ln`. If you still would like to run Powershell, it's available at:

<http://www.microsoft.com/windowsserver2003/technologies/management/powershell/default.mspx>.

Two useful applications can provide commands you will use frequently when running Cake, or any other web service for that matter: Cygwin ([www.cygwin.com](http://www.cygwin.com)) and MinGW ([www.mingw.org](http://www.mingw.org)). Both come with installer applications that are fairly easy to run. (I prefer MinGW running with MSYS, both available through MinGW's web site.)

*Setting permissions in the command line.* Now, to set the necessary permissions for the *tmp* folder, launch your preferred command line program. You will need to point to the path of the *tmp* folder. On the Mac, remember, this is set to the user's *Sites* folder. On the PC, it's the *htdocs* folder in the *c:\xampp* folder.

(Mac)     \$     `cd ~/Sites/cake/app`

(PC)       \$     `cd /c/xampp/htdocs/cake/app`

Use the "chmod" command to set the permissions of the *tmp* folder:

\$     `chmod 777 -R tmp`

And you're done. The *tmp* folder is now ready for Cake.

**Set the `CAKE_SESSION_STRING` in the `core.php` file.** For a web application to communicate with a user's unique set of desktop programs, requests are sent back and forth. The server receives requests and processes them, then provides output back to the user. Frankly, these requests can get interrupted (something you've likely experienced when your browser has a "timeout") or the server can get confused about which request is coming from whom. An essential attribute of all web applications is its use of sessions.

When a session is initialized, the server groups a set of requests together using a session id, a database, or a cookie. Whatever the method, the idea behind the session is that the server can maintain a pseudoconnection with the user, even though the communication could get interrupted along the way. You've run into this when you've logged into your web-based email account or some similar web service. The site application knows that you're logged in and maintains that status until you log out or a certain length of inactivity transpires.

Luckily for us, Cake makes session handling easy. But we do need to make sure that its session string is secure (we wouldn't want users out there to toy with the session handling in an effort to break into our applications).

To do this, open the *app > config > core.php* file and locate line 97, or thereabouts. You'll find a line that looks like this:

```
97 | define('CAKE_SESSION_STRING', 'DYhG93b0qyJfIxfS2guVoUubWwvniR2G0FgaC9mi');
```

This is a PHP definition string. Essentially, this line is saying in PHP to define a variable called "CAKE\_SESSION\_STRING" to have the value "DYhG93b0qyJfIxfS2guVoUubWwvniR2G0FgaC9mi." Because that funky bit of characters comes with Cake, everyone who uses Cake has the same session string. Let's change the second portion, the character string, to something unique.

Web sites like *www.grc.com/passwords.htm* generate alpha-numeric passwords or random strings for better security. Go ahead and grab an encrypted alpha-numeric string, about 40 characters in length, and paste it here. I ended up with:

```
97 | define('CAKE_SESSION_STRING', 'mEayuDrXBhZkdiEJgFzPXvbcBrmKo9CdVGtKyPBr');
```

Now when Cake manages sessions for this application, it will have a more secure string to store. Setting a random string of alpha-numeric characters for the CAKE\_SESSION\_STRING definition is a good idea and worth your typical Cake application setup routine.

**Enter MySQL database connection settings.** Cake needs to know where your databases are located so it can fetch from and insert data into the application's programs. This is done by editing the *app > config > database.php.default* file. You'll need to rename the file to "database.php" (remove the ".default" from the end) and edit it in your plain-text editor of your choice.

```

1 class DATABASE_CONFIG {
2
3     var $default = array(
4         'driver' => 'mysql',
5         'persistent' => false,
6         'host' => 'localhost',
7         'login' => 'user',
8         'password' => 'password',
9         'database' => 'project_name',
10        'prefix' => ''
11    );
12
13    var $test = array(
14        'driver' => 'mysql',
15        'persistent' => false,
16        'host' => 'localhost',
17        'login' => 'user',
18        'password' => 'password',
19        'database' => 'project_name-test',
20        'prefix' => ''
21    );
22 }

```

In this class “DATABASE\_CONFIG,” there are two databases it will connect with: default and test. If you had no intention of creating a separate test database, then you can delete lines 13–21. Plop your database settings into the necessary lines, like so:

```

3     var $default = array(
4         'driver' => 'mysql',
5         'persistent' => false,
6         'host' => 'localhost',
7         'login' => 'root',
8         'password' => 'root',
9         'database' => 'cake',
10        'prefix' => ''
11    );

```

In this tutorial, we’re creating a generic Cake application. In the future, we’ll name the database based on the application we’re building, but for now, we’ll just create a database called “cake.” The

settings above will tell Cake how to connect with this database, but we aren't done yet.... We need to create the database!

**Design your database schema.** It's best to know how the database design will work from the outset. So take some time to get at least a moderate idea of the program you're building first, and then build some tables and fields to fit that design.

This application is super boring, with nothing really in the database. We just want to connect Cake to the database. Fire up the MySQL application of your choice (I'm using CocoaMySQL) and connect to MySQL. Create a database called "cake."

Now that a database actually exists for Cake to connect with, you can go to <http://localhost/cake> in your browser and it will display a new screen:



## CakePHP: the rapid development php framework

Your tmp directory is writable.

Your cache is set up and initialized properly.

FileEngine is being used to cache, to change this edit config/core.php

Settings:

- class: FileEngine
- directory: /Users/dave/Sites/todo/app/tmp/cache/
- prefix: cake\_
- lock: false

Your database configuration file is present.

Cake is able to connect to the database.

Cake is now installed and working correctly. It's time to dive in and start building web apps!

# YOUR FIRST CAKE APP

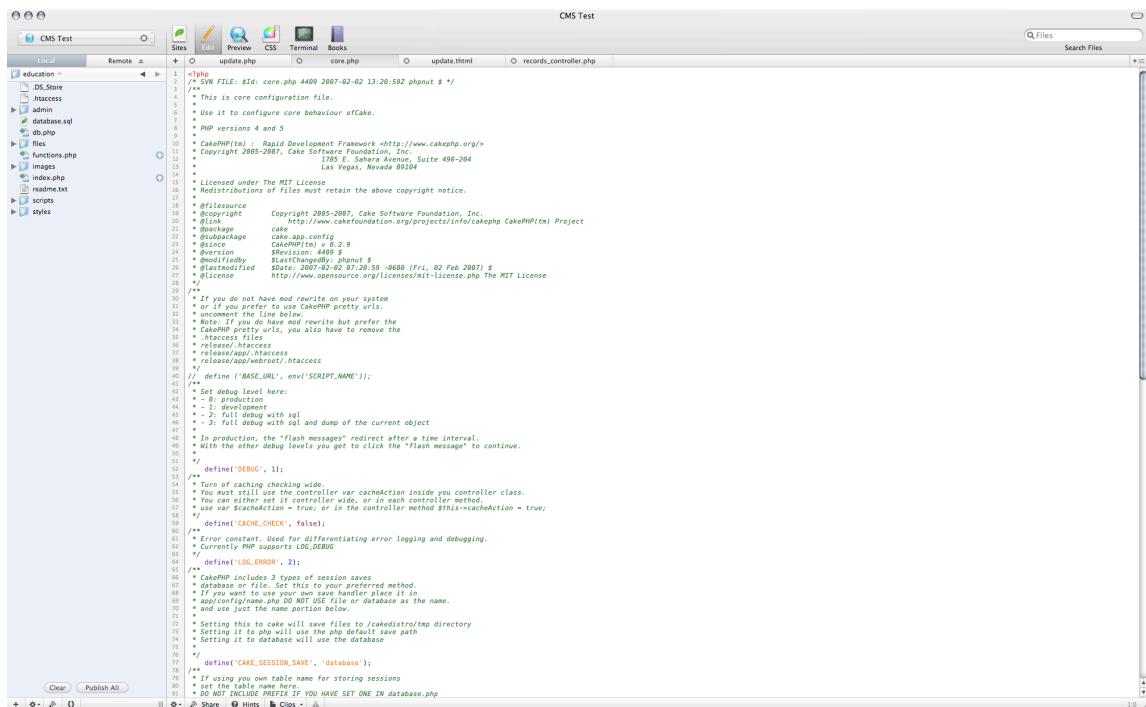
Before we begin building your first Cake app, let's take a moment to discuss important editing options.

## Plain Text Editors and Other Applications

When developing for the web, all the scripts and files are rendered in plain text. It's important that you find a text editor or HTML editing program that fits your workflow style. The available choices may change dramatically from year to year, as many open source projects come and go. Below are some recommendations that will certainly make developing in Cake easier to manage.

### Plain Text Editors

**Coda.** Panic's new web program Coda ([www.panic.com/coda](http://www.panic.com/coda), \$79) is my personal favorite because of how it consolidates almost all aspects of web development into one window.



On the left is a file browser and FTP uploader. A simple click will open any file (local or remote) you see there. In the main text area is where all the code is displayed. Coda supports multiple languages and can colorize your code to help you with your syntax.

It also has a built-in CSS editor, should you need a nice graphical approach to building your style-sheets. And, the terminal is also built into the window, making any command line scripting a breeze. Coda has even more features that will certainly appeal to any Cake developer; it's worth the free demo.

*Disadvantages.* The only disadvantage is that it's only available for the Mac. It doesn't have a MySQL editor; future releases just may come with that built in, like the terminal shell.

**TextMate.** TextMate ([www.macromates.com](http://www.macromates.com), €39) is a lot like Coda, with a file browser and multiple tabs for editing many files at once. One sweet feature is its use of code bundles. Instead of typing monotonous code strings, TextMate come with code bundles that allow you to essentially start a string and it fills in the rest for you. This can cut down on editing time dramatically. Also, you can customize your own code bundles to better match your routines. It, too, comes with a free demo you can download.

*Disadvantages.* Another Mac only program, but well worth its functionality. The price tag, for some, may be a bit steep for what you're getting out of it. In all, TextMate is a good application to try out.

**Adobe Dreamweaver.** Dreamweaver has been a standard HTML application for years ([www.adobe.com/products/dreamweaver](http://www.adobe.com/products/dreamweaver), \$399) and may well continue to be so for more to come. Most of its features are related to web design, not necessarily web development. So, when rendering HTML/CSS and maintaining a web site comprised of static HTML pages, Dreamweaver can be a powerful piece of software. It can support customized templates that you apply to static pages, making editing mass web site assets easier. However, should you want to incorporate PHP code, it is pretty well generic, and you will likely move away from it to a better code editor anyway. The steep price tag is as high as it gets, but well worth it if you've bought the Creative Suite bundle and can combine Dreamweaver with the power of Photoshop and Illustrator (absolute essentials in the graphic design world).

**Nvu** ([www.nvu.com](http://www.nvu.com), free). This open source program is meant to mimmick Dreamweaver and similar WYSIWYG HTML editors. It lacks some of the powerful Ajax features the newest Dreamweaver has, but it well worth the download. For web development, you may want this for your HTML or CSS creation, but may find it a little clumsy for PHP code editing.

**Eclipse.** When it comes to programming in general, Eclipse is an incredibly powerful program ([www.eclipse.org](http://www.eclipse.org), free). The developers community for Eclipse is extensive, making updates to the software efficient and frequent, a bonus for you in the long run. Eclipse comes with different language IDEs, simplifying the editing process by pasting and completing code syntax and strings. Its preview, file browsing, and server setups are also incredibly advantageous, especially if you intend to develop with several other people. Perhaps what sets Eclipse apart from the rest of this list is its CVS and Subversion implementation, making collaboration a cinch. If you have no doubt but that your applications will be developed via the open source community or in a company using some code check-in/check-out system like Subversion, then you must take a look at Eclipse at some point.



## Others

### Other plain text editors.

BEdit ([www.barebones.com/products/bbedit](http://www.barebones.com/products/bbedit), \$125, Mac)

TextWrangler ([www.barebones.com/products/textwrangler](http://www.barebones.com/products/textwrangler), free, Mac)

skEdit ([www.skti.org](http://www.skti.org), \$24.95, Mac)

UltraEdit ([www.ultraedit.com](http://www.ultraedit.com), \$49.95, PC)

### FTP applications.

Transmit ([www.panic.com/transmit](http://www.panic.com/transmit), \$29.95, Mac)

CuteFTP ([www.cuteftp.com](http://www.cuteftp.com), \$39.99, PC)

Fetch ([www.fetchsoftworks.com](http://www.fetchsoftworks.com), \$25, Mac)

### REMINDER

It doesn't matter which applications you use to create your Cake applications. You will need to make sure, though, that you are saving your files in a plain-text format and not another word processing or image format (e.g., a Word .doc file). I'll assume you've picked out a good selection of applications that do all this for you without having to specify a file format, but I will point out which file extensions to use.

## A To-Do List Application

For your first Cake application, let's build a to-do list program. First step: rename the previous application folder from "cake" to "todo." (Whenever we launch the program, just type in <http://localhost/todo>.)

### Create the Database Schema

Our database is the first thing to work with. Remember, we built a database in MySQL already called "cake." Go ahead and rename this if you want, or leave it the same; in either case, in the *app* > *config* > *database.php* file, it's already pointing to the "cake" database and must be changed in the event that you do decide to rename the database.

The database is empty, so we need to create some tables first. This application is simple enough: it's just a list of to-do items that are either completed or not. Let's name the table, then, "items." Next, create the fields. In our to-do list, we'll have the *name* of the item, some *detail* of what needs to be done, whether or not the item is *completed*, a *due date*, and a *priority*. The name field will be the SQL type "varchar" because it will contain only one string of text, not multiple lines. The detail, however, could be a whole paragraph, so we'll assign this field the type "text." The completed field will be either a true or false (in SQL/PHP, we'll use a 1 for true and 0 for false), so make this

field a “tinyint” type. The due date will contain the date and time of the item, and MySQL conveniently has one of those as a type, so this one will be simply of the type “datetime.” When naming this field, you can’t have spaces, so put an underscore between the two words (e.g., “due\_date”). Last, priority is just an integer between 1 and 9; make this “smallint.”

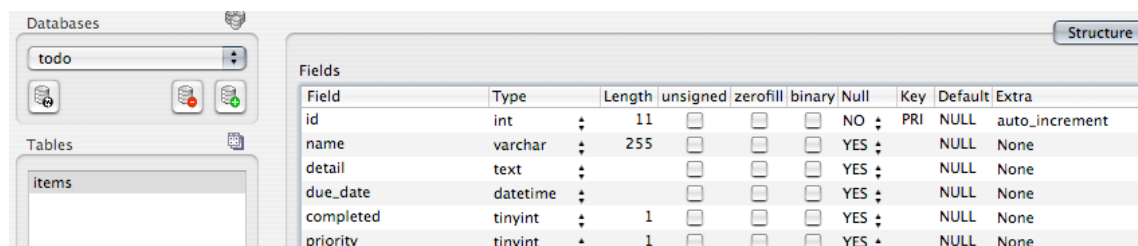
As you create the fields and specify their types, you will be given the option of specifying a field length. In the case of “varchar,” “tinyint,” and “smallint” types, we’ll need to tell MySQL how long the chunk of data to be stored should be. If the user enters more characters than we specify here, MySQL will simply trim off the excess and store up to the field’s limit.

The maximum for “varchar” is 255, which is pretty hefty. Just to be sure, though, let’s plop that value into the name’s field length.

The true/false “tinyint” should be just one character, so make the completed field length 1. For the priority field, set this length to 1 as well.

**The “id” field.** As a standard procedure, you ought to have what’s called a primary key field. This field makes queries faster and helps MySQL to connect records in the database faster. By making an “id” field, each to-do list item stored in the table will have a unique identifier. Cake and MySQL can parse through this more quickly, and Cake is more able to handle the data with a unique id. Almost without exception, every table should have an “id” field, of the type “integer,” and a field length of at least 11. This field will need to be set as the “primary key” with “auto\_increment” added to it. Now, no matter how many items are stored in the database, no two records will have the same id value, a safe precaution to protect against duplication of data and confusing Cake. (Make sure before creating the primary key that you move “id” to the top position—it won’t let you change the field order once the primary key is assigned.)

Your database schema should look something like this:



Field	Type	Length	unsigned	zerofill	binary	Null	Key	Default	Extra
id	int	11	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NO	PRI	NULL	auto_increment
name	varchar	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	YES		NULL	None
detail	text		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	YES		NULL	None
due_date	datetime		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	YES		NULL	None
completed	tinyint	1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	YES		NULL	None
priority	tinyint	1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	YES		NULL	None

## Create the Items Controller

In the `app > controllers` folder, create a new file for the “items” table in the database. Controllers, by default, link up to the table after which they are named. In this case, we have created an “items” table, so the convention in Cake is to name the controller file after this table, using an underscore and the extension “controller.php.” So name this new file “items\_controller.php” and place it in the controllers folder.

Next, in the `items_controller` file, paste the following code:

```

1  <?
2  class ItemsController extends AppController {
3      var $name = 'Items';
4      var $scaffold;
5  }
6  ?>

```

Let me explain what's happening on each line.

One line 1, we are initiating a PHP file. *Important:* depending on your localhost or remote host setup, this line may need to be changed. I'm using, in this example, a type of shorthand for PHP; just the less-than symbol and the question mark. In many setups, this shorthand won't fly, and you'll need to use `<?php` instead. Semantically, shorthand makes your code cleaner and easier to read, so I'll stick with it, but you will want to double-check the examples if you have set up your localhost differently than the earlier tutorial in the chapter "Installing Cake."

Line 2 is necessary for Cake to run the controller. Cake is already starting some of its own PHP code in what it calls the controller. Next, it moves down the tree to the application's own controller, or "AppController." And lastly, it scans the controllers folder for individual controllers. All the way down, we need to tell Cake if we've inserted our own AppController or individual controller file, and we do this by starting a class and extending the previous level of controller. In this case, we created the "ItemsController" class, and it extends out from the "AppController." The braces hold everything, code-wise, that is included in this individual controller.

Line 3 is just good practice. There are a plethora of setup scenarios with which Cake can be installed. By specifying a name for the controller, we bypass potential hazards of Cake not knowing what's what. Technically, you can leave out this line, but it doesn't hurt to habitually include it, and avoid possible bugs.

In line 4, we've called out one of Cake's built-in features: the scaffold. In any controller, you can include this line and Cake will build its own set of HTML forms and tables around what it finds in the database table. In a second, you'll see how helpful this one little string of code can be.

Lines 5 and 6 close out the controller, following PHP's syntax structure.

### *Create the Items Model*

Now, the controller can only do so much. It needs now to be able to connect to the database and filter through the table data we've already set up. Cake separates these functions away from the controller through the use of "models." In every case, the model is the mechanism by which we access the database, check for database errors, and employ any filters to clean up our data.

In the `app > models` folder, create a new file called "item.php." Again, this file naming scheme is a convention used in Cake to make accessing the database more simple. Should you name it something else, you'd need to add more strings of code in your controller file and in this file to tell

Cake where to go. By default, it will search the models folder for a singular term of the database table, in this case, an “item” model.

Paste the following code in the “item.php” file.

```
1 <?
2 class Item extends AppModel {
3     var $name = 'Item';
4 }
5 ?>
```

Things are looking similar to the items\_controller we already created. Line 2 follows the naming convention in Cake, telling Cake that this model extends the AppModel and is called “Item.” We’ve also included the naming string for good measure on line 3. As there is nothing else to do with the data, we’re done, so we close it out with lines 4 and 5.

## Launch Your App

The URL structure of Cake follows a simple pattern: *application/controller/action/parameters*. We’ll examine this flow later on. For right now, to fire up the items controller, type *http://localhost/todo/items* in your web browser.

Thanks to Cake, virtually everything is complete! You’ll see a screen with all the database fields conveniently rendered, a list view in HTML all designed and populated, and the editing, deleting, adding actions all taken care of. Because we told Cake to build the scaffold on line 4 in our items\_controller, it was able to scan the database and create all the typical views that follow CRUD operations (see “CRUD Operations” on page 3).

### Items

Page 1 of 1, showing 0 records out of 0 total, starting on record 0, ending on 0

Id	Name	Detail	Due Date	Completed	Priority	Actions
----	------	--------	----------	-----------	----------	---------

<< previous | next >>

[New Item](#)

## Naming Conventions

By now you can see that much of what Cake does relies on naming conventions. For example, you didn’t need to create a specific connection script that told Cake to connect to the “items” table in the database when the items controller is called. Cake just intuitively knew that the items table was where to go to fetch the data. Well—not exactly! Actually, Cake knew to go there because we followed its *naming convention*. It’s important to understand how naming conventions work in Cake, at this point, and to follow those naming conventions when we build our Cake apps.

## Controllers

When you create a controller, it's important to keep the name of the controller file consistent with the naming convention in Cake. The name will always be the plural form of a record in the database table. Since we are using “items” and each record in the table represents one item, the name of the controller file will follow the name of the table. You will recall when we created the file “items\_controller.php,” I explained why we named it that way. Well, whenever you create a controller file, almost without exception you should follow this convention:

Plural word of the database table followed by an underscore and the word “controller” with the PHP extension (.php).

As long as this is the case, Cake will automatically link the controller up with the model, view, and database records.

## Models

Models are like controllers in their convention—they must follow a specific pattern for Cake to recognize that they're linked up with other controllers and views—but the name is different. The model represents the record in the database table. Each record is singular, so the model name follows this pattern. In our example, each record represent one “item,” so our model file is named “item.php.”

When naming your model files, follow this convention:

Singular form of the database table title followed by the PHP extension (.php).

Cake is pretty good at figuring out irregular pluralities like “university” and “universities” but sometimes it may not put the two together. We'll talk about how to fix this later. For now, just stick with good names that aren't too convoluted, and Cake will bridge the files together as it runs.

## Views

The views are tied to specific functions in your controller file. Therefore, they are named after the function itself. If I make a function called “longtodolist” then I would name the view file responsible for the output of this function “longtodolist.ctp” and store it in a folder called “items” in the *app > views* folder. The name of the folder that holds these individual views corresponds with the name of the controller. In other words, since our controller is plural and links up with the table “items,” the view folder that houses the individual views for this controller will also be named “items.”

## More Than One Word in the Title

If you need to put more than one word in the name of your controller, database table, model, or view, then the convention here is to do so with an underscore:

`big_items_controller.php` (name of controller file)

`big_items` (name of table in the database)

`big_item.php` (name of model file)

`big_items/` (name of views folder)

When in your files, however, the opening class string must be camel-cased for Cake to recognize the script:

```
2 class BigItemsController extends AppController {
3     var $name = 'BigItems';
```

```
2 class BigItem extends AppModel {
3     var $name = 'BigItem';
```

Don't forget to follow these conventions, and Cake will save you a lot of time in connecting files to each other. In reality, the functionality of any framework lives and breathes on these principles of *convention* and consistency.

## Changing the Design

You'll notice here that because we've called out Cake's scaffolding feature, the design uses all of Cake's built-in styles and layout. We can change the overall design easily without affecting the scaffolding feature itself by adjusting the views.

So far the only files we've created were `app > controllers > items_controller.php` and `app > models > item.php`. Since the `items_controller` file contains the `var $scaffold` string, all the views have been dynamically generated. Were this not the case, Cake would go into the `app > views` folder and search for a directory entitled "items" to build the design. We haven't created any yet, and it's using the scaffolding, so Cake is still able to build our application. Let's go ahead and change the design.

### How Views Work

Remember, Cake's framework follows the MVC method: models, views, and controllers. So far, you've become familiar with the model and the controller. Views are separated out from the mix to provide for easier handling of output. Cake organizes all the views in the `app > views` folder.

The views folder contains the following directories:

```
elements/
errors/
helpers/
layouts/
pages/
```

scaffolds/

The layouts folder contains any overall layout files that are wrapped around your application. By creating a file named “default.ctp,” Cake knows to apply this layout instead of its own default scaffolding layout.

**Create the default.ctp file.** In the *app > views > layouts* folder, create a file named “default.ctp,” and put in it your own HTML/CSS code.

## SIDENOTE

In earlier versions of Cake, all view files were given the “.thtml” extension. The developers have chosen to change this to “.ctp” for better system compatibility. Currently, both extensions work, but since .thtml will eventually become deprecated, I’ve chosen to stick with .ctp in the following examples.

Below is a basic, boring, HTML layout to demonstrate how layouts work:

```

1  <html>
2  <head>
3    <title>My Cake To-Do Application</title>
4    <?=$html->css('styles');?>
5  </head>
6  <body>
7    <div id="container">
8      <div id="content">
9        <?=$content_for_layout;?>
10     </div>
11   </div>
12 </body>
13 </html>

```

Lines 1–3 are basic HTML tags that initiate an HTML file and its header, as well as give the page a title, in this case, “My Cake To-Do Application.”

Line 4 is a bit of Cake helper code that will automatically pull in the CSS file named “styles.css” (we’ll create this file in a second).

Line 5 closes the HTML file’s headers.

Lines 6–8 initiate the body space and create two block-level elements with ids “container” and “content.” Lines 10–13 close these tags out.

Line 9 is the key to any layout file in Cake. It’s the string that tells Cake where to put all the views. When you launch any controller that hasn’t specified another layout file, by default Cake will now swap out this line with whatever output you’ve generated through your controllers, models, and views. Because it is a layout file, all the surrounding HTML will be applied to all the output.

**Create a stylesheet for the layout.** Now, in the `app > webroot > css` folder, create a new file named “styles.css.” This file will get pulled into the HTML files that Cake generates through the default layout (because of line 4 in the code sample above). Since it is blank, we shouldn’t see anything flashy whatsoever once the view is rendered.

Now, launch the `items` controller, and you should now see the following screen:

## Items

Page 1 of 1, showing 0 records out of 0 total, starting on record 0, ending on 0

[Id](#)[Name](#)[Detail](#)[Due Date](#)[Completed](#)[Priority](#)[Actions](#)

<< previous

|

next >>

- [New Item](#)

Pretty boring, eh? We can spice this up by simply editing the `app > webroot > css > styles.css` file. In this file, paste the following code, or do something similar:

```
1  * { font-family: "Lucida Grande",Lucida,sans-serif; }
2
3  th {
4      font-size: 14px;
5      font-weight: bold;
6      background-color: #e1e1e1;
7      border-bottom: 1px solid #ccc;
8      padding: 10px;
9  }
10
11 div.actions ul {
12     list-style-type: none;
13 }
```

Line 1, we declare a browser default of the font family. Lines 3–9 contain the styles for the `<th>` tag. And, finally, lines 11–13 make the list items change from a bullet to no bullet type at all. Refresh the items screen, and you’ll see something like this:



## Items

Page 1 of 1, showing 0 records out of 0 total, starting on record 0, ending on 0

<a href="#">Id</a>	<a href="#">Name</a>	<a href="#">Detail</a>	<a href="#">Due Date</a>	<a href="#">Completed</a>	<a href="#">Priority</a>	<a href="#">Actions</a>
<< previous						
next >>						
<a href="#">New Item</a>						

You can see that by placing our own styles into the default layout, we can fully edit the design surrounding our application's output. Any views that are rendered, if they follow the default layout, will also be rendered using the same stylesheet, so there's no need to duplicate styles or designs.

Go back to the `app > views > layouts > default.ctp` file, and change line 4 to the following:

```
4 <?=$html->css('cake.generic');?>
```

When you refresh the items screen, you'll notice that Cake's styles have been implemented without the scaffolding's default titles and such:

### Items

Page 1 of 1, showing 0 records out of 0 total, starting on record 0, ending on 0

<a href="#">Id</a>	<a href="#">Name</a>	<a href="#">Detail</a>	<a href="#">Due Date</a>	<a href="#">Completed</a>	<a href="#">Priority</a>	<a href="#">Actions</a>
--------------------	----------------------	------------------------	--------------------------	---------------------------	--------------------------	-------------------------

<< previous | next >>

[New Item](#)

With all these style changes, the scaffolding is still generating the individual CRUD operation views but now without Cake's built-in layouts.

## Creating Individual Views

What do we do if we want to manipulate the views directly? For example, say you wanted to get rid of the title at the top that reads "List Items" and replace it with "Things I Need To Do." This is where individual views comes in.

Yes, the scaffolding is a nice treasure that makes testing out our code quick and painless, especially if all we want to do is get to playing around with our database to make sure it's working right. But it can't possibly read our minds, and it can only create some generic views. To add or subtract from the application output will require us to manually build such outputs. No worries—this, too, is made much easier through the use of the framework.

## Add the Index Action to the Controller

Open up the `app > controllers > items_controller.php` file. Starting at line 4, paste in the following code:

```
4   var $scaffold;
5
6   function index() {
7       $this->set('items', $this->Item->findAll());
8   }
```

Here's what's happening in the controller. Line 4 still has the scaffolding callout string, so by default, if Cake can't find a function in this file to match an action typed by the user (e.g., `localhost/todo/items/edit` is typed in the URL, but there is no "edit" function in this file), then it will default to building the scaffold.

Lines 6–8 contain a new function called `index`. We're now creating this function because it is the default action for every controller. For example, when I type "`http://localhost/todo/items/`", Cake searches for a function in the `items_controller` called "index." It's the same as if I type "`http://localhost/todo/items/index`". Previously, we hadn't specified the index action in the file, so Cake used its pre-built scaffolding functions to render the index view.

Now, though, lines 6–8 are specifically telling Cake how to handle the index action request. Cake will not use the scaffolding here since it will find this function.

On line 7, we're setting a new variable to be used in the view file (which we haven't created yet) called "items" and the value we're giving this variable is `$this->Item->findAll()`. Essentially, this line is telling the model to go into the database, locate the "items" table, and find all records that are in that table. To summarize, then, this line is telling Cake to fetch all the data in the "items" table in the database and slap it all into a variable called "items" which we will use in the view file.

Line 8 closes out the function.

Now, when you refresh the items controller screen, you should see an error page like so:

### Missing view

**You are seeing this error because the view for `ItemsController::index()`, could not be found.**

**Notice:** If you want to customize this error message, create `app/views/errors/missing_view.ctp`

**Fatal:** Confirm you have created the file: `/Users/dave/Sites/todo/app/views/items/index.ctp`

What just happened is that Cake effectively located the index function you created, but was unable to find the corresponding view file. In the second gray box, you'll see this error message:

Fatal: Confirm you have created the file :  
/Users/dave/Sites/todo/app/views/items/index.ctp

Cake is telling you that it looked in the *app > views* folder for a folder called “items” with a file called “index.ctp” in it and couldn’t find it.

That’s cause we haven’t created it yet!

## REMINDER

Whenever you get missing view errors like this one, Cake simply cannot locate a view file to correspond with a function you’ve created. You can check your controller file to make sure that the function is spelled correctly or check your views folder to make sure the nested folder or file are spelled right.

### *Create the Index View*

In the *app > views* folder, create a new folder called “items.” This is so that any individual view files that need to be rendered for the items controller will be fetched correctly.

Next, create a file named “index.ctp” in the *app > views > items* folder.

Insert the following code into the new index.ctp file:

```

1 <h1>Things I Need To Do</h1>
2 <table class="inav" cellpadding="0" cellspacing="0">
3 <thead>
4 <tr>
5 <th>Id</th>
6 <th>Name</th>
7 <th>Detail</th>
8 <th>Due Date</th>
9 <th>Completed</th>
10 <th>Priority</th>
11 </tr>
12 </thead>
13 <tbody>
14 <? foreach ($items as $item): ?>
15 <tr>
16 <td><?=$item['Item']['id'];?></td>
17 <td><?=$item['Item']['name'];?></td>
18 <td><?=$item['Item']['detail'];?></td>
19 <td><?=$item['Item']['due_date'];?></td>
20 <td><?=$item['Item']['completed'];?></td>
21 <td><?=$item['Item']['priority'];?></td>
22 </tr>
23 <? endforeach; ?>
24 </tbody>
25 </table>
26 <div class="actions">
27 <ul>
28 <li><?=$html->link('New Item','/items/add');?></li>
29 </ul>
30 </div>

```

Essentially, what we have done here is recreate the scaffolding view (assuming you're still using the "cake.generic.css" file instead of your own). But now that the code is right in front of us, we can toy with any aspect of it and customize it to meet our own wants. Once you've refreshed the items controller, you'll see this:

## Things I Need To Do

Id	Name	Detail	Due Date	Completed	Priority
<a href="#">New Item</a>					

Now, instead of seeing an `<h1>` title in the view that says the generic “List Items,” it reads “Things I Need To Do.” Let me explain what has happened, line by line, in your `index.ctp` view file.

Line 1 is where the change happened with the “Things I Need To Do” header. (I put this in there to show some measure of proof that we are not in the scaffolding environment, even though it’s looking almost identical.)

Lines 2–13 are the HTML strings that create the same table as the scaffolding, with its header cells.

Lines 14–23 are a loop through the variable we created in the *controllers > items\_controller.php* file on line 7. Line 14 starts the loop by saying, essentially, for each record from the database that is stored in the variable “items,” create a new variable called “item.” Line 23 marks the ending point for the loop. So everything between lines 14 and 23 will repeat for each record in the database that has been stored and brought into the “items” variable. You can see that I’ve set up a table row for each “item” and that there are corresponding table cells (lines 16–21) that pull out each field’s data in the database.

Lines 26–30 create an unordered list with the link to create a new to-do list item. On line 27, there is a little bit of Cake helper code that generates links. Similar to the CSS helper string in the default.ctp layout file, this line tells Cake to generate the HTML for us, which is nice; no matter where we may move or install this application, as long as Cake is working, the link to create a new to-do item will be unbroken. (We’ll discuss Cake’s snazzy set of HTML helpers later.)

Because we haven’t created any to-do list items in the database, we don’t see this effect really take place; the `$items` variable is empty. Click “New Item,” create a new to-do list item, and watch Cake pull that entry into the newly customized index view.

## Using Bake to Create Views

We’ve walked through how to manually create the index view. Actually, though, you should be able to get around having to type these basic CRUD functions by hand. Included with Cake is a handy console script called the Bake script (it’s found in the *cake > console > libs* folder, the *bake.php* script). Not only will it save you tons of time in generated the needed code to build these views, it will also show you some basic Cake code that will help you understand how Cake makes use of models, views, and controllers.

### Getting the Bake Script Working In Our Localhost

Because we have used either MAMP or XAMPP for our localhost setup, we’ll need to configure the command line to work with the Bake script (see “Setup Routines,” pages 19–20 for instructions on how to get started with the command line).

**Fix the .profile file.** The command line console will use a file, usually invisible in your OS, named “.profile” when it executes commands. Fortunately, we can add some of our own customized environment settings to tell the console where to go when we execute Bake commands.

There are a number of ways to open the .profile file. You can use the terminal command

```
$ vi .profile
```

to edit .profile in the console itself. Or,

```
$ open .profile
```

will launch the .profile file in whatever your default file editor is. However, if you're like me, you'd probably rather edit this file in a simple plain-text editor. I like to use Panic's Coda to open the file directly in its plain-text editor window. A simple search on Version Tracker ([www.versiontracker.com](http://www.versiontracker.com)) or Mac Update ([www.macupdate.com](http://www.macupdate.com)) will help you find a plain-text editor and invisible file viewer for both Mac and PC.

On the Mac, the needed .profile file is housed in the user's home directory (e.g., `/Users/dave/.profile`). On the PC, it will depend on what console application you're running (e.g., `C:\msys\1.0\etc\profile`).

However you decide to edit the .profile file, be sure to add the corresponding line listed below, depending on your localhost setup (one line, no returns).

*MAMP on a Mac.*

```
alias cake="/applications/mamp/bin/php5/bin/php ~/Sites/todo/cake/console/cake.php"
```

*XAMPP on a Mac.*

```
alias cake="/applications/xampp/xamppfiles/bin/php ~/Sites/todo/cake/console/cake.php"
```

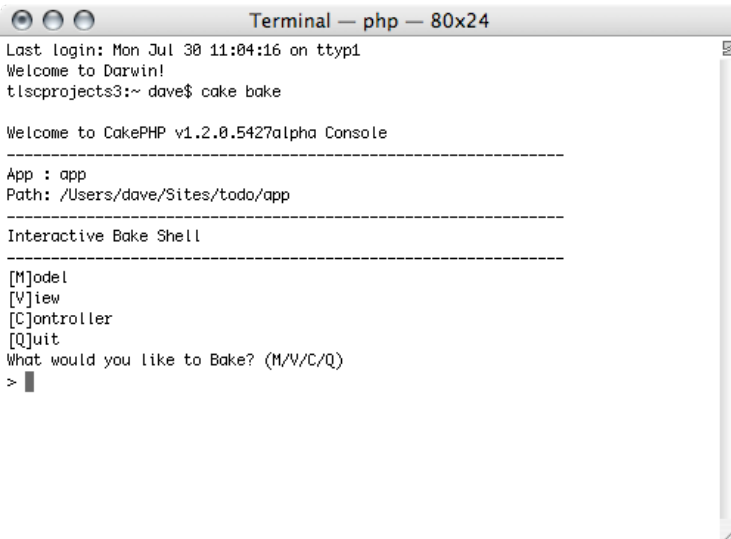
*XAMPP on a PC.*

```
alias cake="/c/xampp/php/php4/phpcli.exe /c/xampp/htdocs/todo/cake/console/cake.php"
```

Next, open the terminal and type:

```
$ cake bake
```

If all runs well, you should get a screen like so:



```

Terminal — php — 80x24
Last login: Mon Jul 30 11:04:16 on ttty1
Welcome to Darwin!
tiscprojects3:~ dave$ cake bake

Welcome to CakePHP v1.2.0.5427alpha Console
-----
App : app
Path: /Users/dave/Sites/todo/app
-----
Interactive Bake Shell
-----
[M]odel
[V]iew
[C]ontroller
[Q]uit
What would you like to Bake? (M/V/C/Q)
> █

```

## Bake the CRUD Views

Let's Bake the CRUD operations' views using the Bake script. First, delete the *app > views > items* folder with its *index.ctp* file. Next, go into the *app > controllers > items\_controller.php* file and zap the *index* function.

Open the terminal/console application of your choice and start the Bake program like so:

```
$ cake bake
```

It will ask you what to bake:

```
What would you like to Bake? (M/V/C/Q)
```

Here, it's asking if you want to bake a model, view, or controller, or quit. Well, we've already made the model and the controller for "items," so we'll tell it to bake the view for us. Type "v" and hit enter.

Bake will now go into the database and find all the tables there and list them numerically. Since we have only created one table ("items"), we'll type the number 1 and hit enter.

It will ask you if you want to bake the view interactively. Hit yes to see each step of the view baking. (If you select "no," then it will overwrite existing view files; fortunately, we've already removed those.)

```
Would you like to create some scaffolded views (index, add, view, edit) for this controller?
```

This is what we're looking for. Here, Bake will go through the scaffold for our controller and model and put together all the code it would use "behind the scenes" but place it in our controller and view files. Hit "yes" to proceed. It will ask if you want to create views for admin routing. Type "n" for no, and hit enter.

View Scaffolding Complete.

Going into your *app > views* folder, you'll find a new folder named "items" and in it will be four files built by Bake:

```
add.ctp
edit.ctp
index.ctp
view.ctp
```

These files have all the necessary code and HTML markup to render the basic CRUD functions like Cake's scaffolding. But we still need to supply the needed controller functions as well.

Start the Bake program again. Tell it you want to bake a controller. Like before, it will ask you for which number table in the database from which to build the controller. Select the items table and continue. Choose "yes" when asked if you want to proceed interactively, and when it asks you if you want scaffolding, choose "no."

```
Would you like to include some basic class methods (index(), add(), edit(), view())?
```

Yes. This will create the controller file with the code to render the index, add, edit, and view views. Since those view files are already created and in the *app > views > items* folder, these functions will link up with them, and the application will run just like the scaffolding, except you'll then be able to edit the code.

Select no when it asks you if you want admin routing or to use a different model. Continue selecting "no," except for when it asks

```
Would you like to use Sessions? (y/n)
```

and choose "yes" here. It will ask if you want to overwrite the previous "items\_controller.php" file. Enter yes. Enter no when asked if you want to bake the unit test files.

Open up the *app > controllers > items\_controller.php* file, and you should now see 64 lines of new code.



```

1  <?php
2  class ItemsController extends AppController {
3
4      var $name = 'Items';
5      var $helpers = array('Html', 'Form' );
6
7      function index() {
8          $this->Item->recursive = 0;
9          $this->set('items', $this->paginate());
10     }
11
12     function view($id = null) {
13         if (!$id) {
14             $this->Session->setFlash('Invalid Item.');
```

```

15             $this->redirect(array('action'=>'index'), null, true);
16         }
17         $this->set('item', $this->Item->read(null, $id));
18     }
19
20     function add() {
21         if (!empty($this->data)) {
22             $this->cleanUpFields();
23             $this->Item->create();
24             if ($this->Item->save($this->data)) {
25                 $this->Session->setFlash('The Item has been saved');
```

```

26                 $this->redirect(array('action'=>'index'), null, true);
27             } else {
28                 $this->Session->setFlash('The Item could not be saved. Please, try again.');
```

```

29             }
30         }
31     }
32
33     function edit($id = null) {
34         if (!$id && empty($this->data)) {
35             $this->Session->setFlash('Invalid Item');
```

```

36             $this->redirect(array('action'=>'index'), null, true);
37         }
38         if (!empty($this->data)) {
```

```

39     $this->cleanUpFields();
40     if ($this->Item->save($this->data)) {
41         $this->Session->setFlash('The Item saved');
42         $this->redirect(array('action'=>'index'), null, true);
43     } else {
44         $this->Session->setFlash('The Item could not be saved. Please, try again.');
```

```

45     }
46 }
47 if (empty($this->data)) {
48     $this->data = $this->Item->read(null, $id);
49 }
50 }
51
52 function delete($id = null) {
53     if (!$id) {
54         $this->Session->setFlash('Invalid id for Item');
55         $this->redirect(array('action'=>'index'), null, true);
56     }
57     if ($this->Item->del($id)) {
58         $this->Session->setFlash('Item #'.$id.' deleted');
59         $this->redirect(array('action'=>'index'), null, true);
60     }
61 }
62
63 }
64 ?>
```

## ***Cleaning Up the Views***

You may have noticed, if you tinkered around at all with the application so far, that there may be some clunky problems once you try to create and edit to-do list items.

- Priority is an open field, not a select menu with a number value.
- In the index view, a 1 or 0 is displayed instead of a “Yes” or “No.”

Since Bake has given us all the code we need to play with, rather than build each line, we can just edit the parts that need fixing.

## Make the Priority Field a `<select>` Tag

Right now, because Priority is stored as a smallint field type in the database, Cake is providing an open text box. It would be better, given the nature of our application, to create a select field with predetermined values.

To change this, go back to the *app > views > items > add.ctp* and locate line 10.

```
10 | echo $form->input('priority');
```

By now you should be able to spot the problem: line 10 has the Cake Form helper creating an input field instead of a select menu. To make the change, replace line 10 with:

```
10 | echo $form->select('priority',array('1'=>'1','2'=>'2','3'=>'3'));
```

Here, we've inserted the Form helper's `select` builder, called it "priority" for proper saving, and an array of numbers and values. You could extend this list to nine values by continuing the array following the provided sequence.

Next, the *app > views > items > edit.ctp* file needs to be fixed as well. This file is virtually identical to the *add.ctp* file, so the change is just as simple. Line 11 in this file are identical to line 10 in the *add.ctp* file. Make the exact same change by replacing line 11 with the same string we used in the *add.ctp* example.

Now with a select menu provided, our fix is complete!

## Change the Index View to Better Display the Completed Field

The index view shows a 1 or 0 value for the completed field. It would make more sense to have this read "yes" or "no" instead. To make this change, open up the *app > views > items > index.ctp* file and locate lines 34–36:

```
34 | <td>
35 |     <?php echo $item['Item']['completed']?>
36 | </td>
```

The problem here isn't that Cake is confusing the field; it's that it is actually showing the value itself rather than interpreting the value. We have to interpret it for better functionality. A simple `if/then` statement will fix this problem.

**PHP's ternary operator.** A simple `if/then` statement is incredibly common in computer programming, no matter what language you're using. The ternary operator provides some important shorthand to make for cleaner code.

Without the ternary operator, the code would look like this:

```
if ($item['Item']['completed']==1) {
    echo "Yes";
}
```

```

    } else {
        echo "No";
    }

```

With the ternary operator, we can cut down on extra characters nicely:

```
echo ($item['Item']['completed']==1 ? "Yes" : "No");
```

What we have done is house the if/then statement between parentheses. The “if” chunk of logic comes before the question mark and the “else” is designated by the colon. The `echo` command comes before the parentheses because no matter what the if/then statement encounters, it will output some kind of echo string.

In Cake, ternary operators can make your views much easier to read and render. In this case, instead of plopping around 5 lines of code into our view, we can maintain a one-line change.

So, on line 35, replace it with our ternary operator:

```
35 | <?php echo ($item['Item']['completed']==1 ? "Yes" : "No");?>
```

Launch the index view and notice that all your to-do list items have a “Yes” or “No” for the completed field. Now that we’ve made the changes, our to-do list application is finished.

## Important Points

In this chapter, we created a basic Cake application using MySQL, scaffolding, and Bake. By now, you should be able to—

- Setup a localhost environment
- Install Cake
- Create a MySQL database
- Create a new Cake application
- Configure a Cake application to use the database
- Create controllers, models, and views
- Change the default layout of the application
- Use the scaffolding with database tables
- Configure and run the Bake script
- Edit controllers, models, and views generated by the Bake script

In building more advanced Cake applications, not much is different from this general routine, so it’s worthwhile to practice the operations outlined in this chapter until they’re pretty easy to do on your own.

Frameworks are designed to work in this fashion: start small, then add one piece at a time. I suppose that's the same basic philosophy of all programming, except with a framework, you start out with a lot of code already created for you, then branch out from there. As much as possible, try to let Cake do operations for you, then build out from what it gives you. You'll find that you truly can build applications rapidly that are every bit as powerful as enterprise software.

In practicing the scenario used in this chapter, try increasing your speed rather than improving every little bit of functionality. At this point, if you can build this to-do list application in under ten minutes, you'll be well-prepared for more advanced Cake application scenarios.